Computational Grand Challenges in Assembling the Tree of Life: Problems and Solutions

DAVID A. BADER

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
USA

USMAN ROSHAN

Computer Science Department
New Jersey Institute of Technology
Newark, NJ 07102
USA

ALEXANDROS STAMATAKIS

Institute of Computer Science
Foundation for Research and Technology-Hellas
Heraklion, Crete
GR-711 10 Greece

Abstract

The computation of ever larger as well as more accurate phylogenetic (evolutionary) trees with the ultimate goal to compute the tree of life represents one of the grand challenges in High Performance Computing (HPC) Bioinformatics. Unfortunately, the size of trees which can be computed in reasonable time based on elaborate evolutionary models is limited by the severe computational cost inherent to these methods. There exist two orthogonal research directions to overcome this challenging computational burden: First, the development of novel, faster, and more accurate heuristic algorithms and second, the application of high performance computing techniques. The goal of this chapter is to provide a comprehensive introduction to the field of computational evolutionary

ISSN: 0065-2458/DOI: 10.1016/S0065-2458(06)68004-2

biology to an audience with computing background, interested in participating in research and/or commercial applications of this field. Moreover, we will cover leading-edge technical and algorithmic developments in the field and discuss open problems and potential solutions.

1. Phylogenetic Tree Reconstruction

In this section, we provide an example of B&B applied to reconstructing an evolutionary history (phylogenetic tree). Specifically, we focus on the shared-memory parallelization of the maximum parsimony (MP) problem using B&B based on work by Bader and Yan [1–4].

1.1 Biological Significance and Background

All biological disciplines agree that species share a common history. The genealogical history of life is called phylogeny or an evolutionary tree. Reconstructing phylogenies is a fundamental problem in biological, medical, and pharmaceutical research and one of the key tools in understanding evolution. Problems related to phylogeny reconstruction are widely studied. Most have been proven or are believed to be NP-hard problems that can take years to solve on realistic datasets [5,6]. Many biologists throughout the world compute phylogenies involving weeks or years of

computation without necessarily finding global optima. Certainly more such computational analyses will be needed for larger datasets. The enormous computational demands in terms of time and storage for solving phylogenetic problems can only be met through high-performance computing (in this example, large-scale B&B techniques).

A phylogeny (phylogenetic tree) is usually a rooted or unrooted bifurcating tree with leaves labeled with species, or more precisely with taxonomic units (called *taxa*) that distinguish species [7]. Locating the root of the evolutionary tree is scientifically difficult so a reconstruction method only recovers the topology of the unrooted tree. Reconstruction of a phylogenetic tree is a statistical inference of a true phylogenetic tree, which is unknown. There are many methods to reconstruct phylogenetic trees from molecular data [8]. Common methods are classified into two major groups: criteria-based and direct methods. Criteria-based approaches assign a score to each phylogenetic tree according to some criteria (e.g., parsimony, likelihood). Sometimes computing the score requires auxiliary computation (e.g. computing hypothetical ancestors for a leaf-labeled tree topology). These methods then search the space of trees (by enumeration or adaptation) using the evaluation method to select the best one. Direct methods build the search for the tree into the algorithm, thus returning a unique final topology automatically.

We represent species with binary sequences corresponding to morphological (e.g. observable) data. Each bit corresponds to a feature, call a *character*. If a species has a given feature, the corresponding bit is one; otherwise, it is zero. Species can also be described by molecular sequence (nucleotide, DNA, amino acid, protein). Regardless of the type of sequence data, one can use the same parsimony phylogeny reconstruction methods. The evolution of sequences is studied under a simplifying assumption that each site evolves independently.

The Maximum Parsimony (MP) objective selects the tree with the smallest total evolutionary change. The *edit distance* between two species as the minimum number of evolutionary events through which one species evolves into the other. Given a tree in which each node is labeled by a species, the *cost* of this tree (tree length) is the sum of the costs of its edges. The cost of an edge is the edit distance between the species at the edge endpoints. The *length* of a tree T with all leaves labeled by taxa is the minimum cost over all possible labelings of the internal nodes.

Distance-based direct methods [9–11] require a distance matrix D where element d_{ij} is an estimated evolutionary distance between species i and species j. The distance-based Neighbor-Joining (NJ) method quickly computes an approximation to the shortest tree. This can generate a good early incumbent for B&B. The neighbor-joining (NJ) algorithm by Saitou and Nei [12], adjusted by Studier and Keppler [13], runs in $O(n^3)$ time, where n is the number of species (leaves). Experimental work shows that the trees it constructs are reasonably close to "true" evolution

of synthetic examples, as long as the rate of evolution is neither too low nor too high. The NJ algorithm begins with each species in its own subtree. Using the distance matrix, NJ repeatedly picks two subtrees and merge them. Implicitly the two trees become children of a new node that contains an artificial taxon that mimics the distances to the subtrees. The algorithm uses this new taxon as a representative for the new tree. Thus in each iteration, the number of subtrees decrements by one till there are only two left. This creates a binary topology. A distance matrix is *additive* if there exists a tree for which the inter-species tree distances match the matrix distances exactly. NJ can recover the tree for additive matrices, but in practice distance matrices are rarely additive. Experimental results show that on reasonable-length sequences parsimony-based methods are almost always more accurate (on synthetic data with known evolution) than neighbor-joining and some other competitors, even under adverse conditions [14]. In practice MP works well, and its results are often hard to beat.

In this section we focus on reconstructing phylogeny using maximum parsimony (minimum evolution). A brute-force approach for maximum parsimony examines all possible tree topologies to return one that shows the smallest amount of total evolutionary change. The number of unrooted binary trees on n leaves (representing the species or taxa) is $(2n-5)!! = (2n-5)\cdot(2n-7)\cdots 3$. For instance, this means that there are about 13 billion different trees for an input of $n = 13$ species. Hence it is very time-consuming to examine all trees to obtain the optimal tree. Most researchers focus on heuristic algorithms that examine a much smaller set of most promising topologies and choose the best one examined. One advantage of B&B is that it provides instance-specific lower bounds, showing how close a solution is to optimal [15].

The phylogeny reconstruction problem with maximum parsimony (MP) is defined as follows. The input is a set of c characters and a set of taxa represented as length-c sequences of values (one for each character). For example, the input could come from an aligned set of DNA sequences (corresponding elements matched in order, with gaps). The output is an unrooted binary tree with the given taxa at leaves and assignments to the length-c internal sequences such the resulting tree has minimum total cost (evolutionary change). The characters need not be binary, but each usually has a bounded number of states. Parsimony criteria (restrictions on the changes between adjacent nodes) are often classified into Fitch, Wagner, Dollo, and Generalized (Sankoff) Parsimony [7]. In this example, we use the simplest criteria, Fitch parsimony [16], which imposes no constraints on permissible character state changes. The optimization techniques we discuss are similar across all of these types of parsimony.

Given a topology with leaf labels, we can compute the optimal internal labels for that topology in linear time per character. Consider a single character. In a leaf-to-root sweep, we compute for each internal node v a set of labels optimal for the

subtree rooted at v (called the Farris Interval). Specifically, this is the intersection of its children's sets (connect children though v) or, if this intersection is empty, the union of its children's sets (agree with one child). At the root, we choose an optimal label and pass it down. Children agree with their parent if possible. Because we assume each site evolves independently, we can set all characters simultaneously. Thus for m character and n sequences, this takes $O(nm)$ time. Since most computers can perform efficient bitwise logical operations, we use the binary encoding of a state in order to implement intersection and union efficiently using bitwise AND and bitwise OR. Even so, this operation dominates the parsimony B&B computation.

The following sections outline the parallel B&B strategy for MP that is used in the GRAPPA (Genome Rearrangement Analysis through Parsimony and other Phylogenetic Algorithms) toolkit [2]. Note that the maximum parsimony problem is actually a minimization problem.

1.2 Strategy

We now define the *branch*, *bound*, and *candidate* functions for phylogeny reconstruction B&B. Each node in the B&B tree is associated with either a partial tree or a complete tree. A tree containing all n taxa is a *complete tree*. A tree on the first k ($k < n$) taxa is a *partial tree*. A complete tree is a candidate solution. Tree T is *consistent* with tree T' iff T can be reduced into T'; i.e., T' can be obtained from T by removing all the taxa in T that are not in T'. The subproblem for a node with partial tree T is to find the most parsimonious complete tree consistent with T.

We partition the frontier into *levels*, such that level k, for $3 \leqslant k \leqslant n$, represents the candidates (i.e., partial trees when $k < n$) containing the first k taxa from the input. The root node that contains the first three taxa (hence, indexed by level 3) since there is only one possible tree topology with three leaves. The branch function finds the immediate successors of a node associated with a partial tree T_k at level k by inserting the $(k+1)$st taxon at any of the $2k-3$ possible places. A new node (with this taxon attached) can join in the middle of any of the $2k-4$ edges not adjacent to the root or anywhere on the path through the root. For example, in Fig. 1, the root on three taxa is labeled (A), its three children at level four are labeled (B), (C), and (D), and a few trees at level five (labeled (1) through (5)) are shown. The search space explored by this approach depends on the addition order of taxa, which also influences the efficiency of the B&B algorithm. This issue is important, but not further addressed in this chapter.

We use depth-first search (DFS) as our primary B&B search strategy, and a heuristic best-first search (BeFS) to break ties between nodes at the same depth.

Next we discuss the bound function for maximum parsimony. A node v associated with tree T_k represents the subproblem to find the most parsimonious tree in the

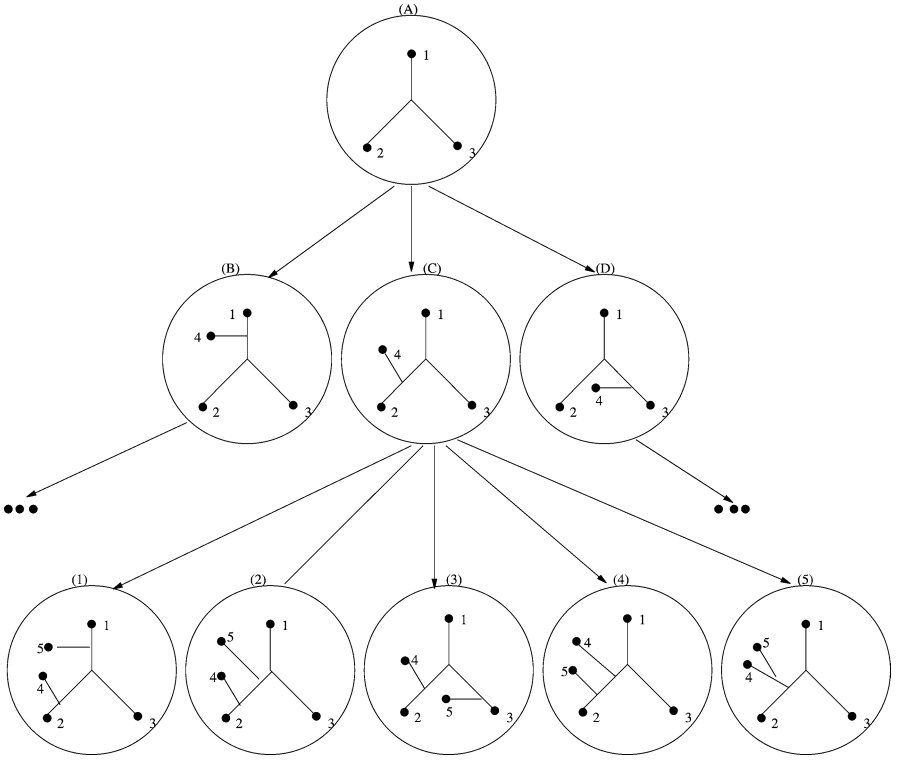

FIG. 1. Maximum Parsimony B&B search space.

search space that is consistent with T_k. Assume T_k is a tree with leaves labeled by $S_1, \ldots, S_k$. Our goal is to find a tight lower bound of the subproblem. However, one must balance the quality of the lower bound against the time required to compute it in order to gain the best performance of the overall B&B algorithm.

Hendy and Penny [15] describe two practical B&B algorithms for phylogeny reconstruction from sequence data that use the cost of the associated partial tree as the lower bound of this subproblem. This traditional approach is straightforward, and obviously, it satisfies the necessary properties of the bound function. However, it is not tight and does not prune the search space efficiently. Purdom et al. [17] use single-character discrepancies of the partial tree as the bound function. For each character one computes a difference set, the set of character states that do not occur among the taxa in the partial tree and hence only occur among the remaining taxa. The single-character discrepancy is the sum over all characters of the number of the elements

in these difference sets. The lower bound is therefore the sum of the single-character discrepancy plus the cost of the partial tree. This method usually produces much better bounds than Hendy and Penny's method, and experiments show that it usually fathoms more of the search space [17]. Another advantage of Purdom's approach is that given an addition order of taxa, there is only one single-character discrepancy calculation per level. The time needed to compute the bound function is negligible.

Next we discuss the candidate function and incumbent x_I. In phylogeny reconstruction, it is expensive to compute a meaningful feasible solution for each partial tree, so instead we compute the upper bound of the input using a direct method such as neighbor-joining [12,13] before starting the B&B search. We call this value the global upper bound, $f(x_I)$, the incumbent's objective function. In our implementation, the first incumbent is the best returned by any of several heuristic methods.

The greedy algorithm [18], an alternative incumbent heuristic, proceeds as follows. Begin with a three-taxa core tree and iteratively add one taxon at a time. For an iteration with an k-leaf tree, try each of the $n - k$ remaining taxon in each of the $2k - 2$ possible places. Select the lowest-cost $(k + 1)$-leaf tree so formed.

Adding taxa in a different order yields different trees. Swofford and Begle [20] describe various ways of producing the provisional MP tree considering the order of taxon addition. In the *as is* method, the initial core tree is produced by the first three taxa given in the data set and the following taxon addition is done according to the taxon order in the data set. In the *random* method, pseudo-random numbers are used to determine the order of taxon addition.

Any program, regardless of the algorithms, requires implementation on a suitable data structure. As mentioned previously, we use DFS as the primary search strategy and BeFS as the secondary search strategy. For phylogeny reconstruction with n taxa, the depth of the subproblems ranges from 3 to n. So we use an array to keep the open subproblems sorted by DFS depth. The array element at location i contains a priority queue (PQ) of the subproblems with depth i, and each item of the PQ contains an external pointer to stored subproblem information.

CAP This is nice, but it seems like a lot of information on to reach the conclusion that this does not matter. I have pulled some pieces out into the discussion.

There are many ways to organize a PQ (see [19] for an overview). From an algorithmic point of view, a B&B algorithm consists of carrying out a series of basic operations on a set of nodes with different or equal priorities: *deletemin* (select and delete the highest priority element), *insert* (insert a new element with a predefined priority), and *delete greater* (delete elements with higher priority higher than a given value). In the literature, PQs are usually represented by heaps, in which each item always has a higher priority than its children. There exists a great variety of algorithms that manage a heap: D-heap [**?**], Leftist-heap [**?**], Skew-heap [**?**], Binomial queue [**?**], Pairing heap [**?**]. The oldest and most popular heap is the D-heap as used

in heap sort. In a D-heap, the tree is embedded in an array, using the rule that the first location holds the root of the tree, and the locations $2i$ and $2i + 1$ are the children of location i. Serial experimental results [**?**,**?**] show that the D-heap in a B&B does not yield an efficient PQ implementation and that a skew-heap is one of the most efficient serial algorithms for heap implementation. The skew-heap is self-adjusting version of the Leftist-heap, and the operations on the skew-heap use a heuristic to keep tree balanced.

Several PQs which are not a heap structure have also been proposed, for example, the Splay-tree [**?**,**?**] and the funnels [**?**] (table and tree). The funnels use the fact that in a given combinatorial optimization problem, priorities lie in a small given interval $[1, \ldots, S]$.

The priority queues (PQs) support best-first-search tie breaking and allow efficient deletion of all dominated subproblems whenever we find a new incumbent. There are many ways to organize a PQ (see [19] for an overview). In the phylogeny reconstruction problem, most of the time is spent evaluating the tree length of a partial tree. The choice of PQ data structures does not make a significant difference. So for simplicity, we use a D-heap for our priority queues. A heap is a tree where each node has higher priority than any of its children. In a D-heap, the tree is embedded in an array. The first location holds the root of the tree, and locations $2i$ and $2i + 1$ are the children of location i.

1.3 Parallel Framework

Our parallel maximum parsimony B&B algorithm uses shared-memory. The processors can concurrently evaluate open nodes, frequently with linear speedup. CAP—this is an issue for all parallel systems, moved forward to general B&B section.

Second, a shared-memory platform makes available a large, shared memory that can hold shared data structures, such as the arrays of priority queues representing the frontier. For example, one of the largest SMP systems to date, the IBM pSeries 690, uses 32 Power4+ 1.9 GHz microprocessors and one terabyte of global memory in its largest configuration. Thus, the data structures representing the search space and incumbent can be shared (concurrently accessed by the processors) with little overhead, and the sheer amount of main memory allows for a tremendous number of active frontier nodes to be saved for later exploration, rather than sequential approaches that often store only a small number of frontier nodes due to space limitations and throw away other nodes that do not seem promising at the time (but may contain the optimal tree). As described in Section 1.2, for each level of the search tree (illustrated in Fig. 1), we use a priority queue represented by binary heaps

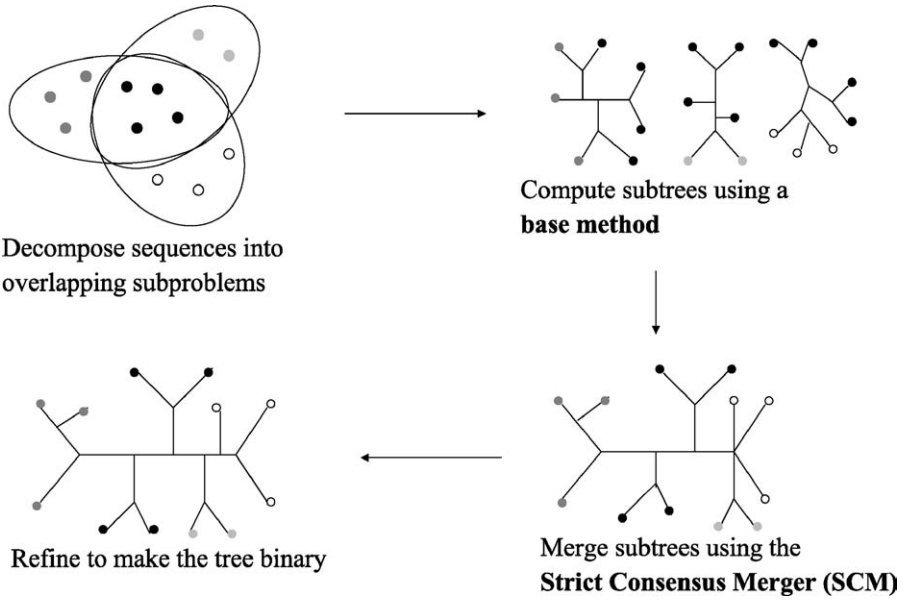

FIG. 2. The array of priority queues, one for each level in the Maximum Parsimony B&B search space.

to maintain the active nodes in a heuristic order. The processors concurrently access these heaps. To ensure each subproblem is processed by exactly one processor and to ensure that the heaps are always in a consistent state, at most one processor can access any part of a heap at once. Each heap H_i (at level i) is protected by a lock $Lock_i$. Each processor locks the entire heap H_i whenever it makes an operation on H_i.

We use a heap H_i for each level i to save the active frontier nodes at that level (see Fig. 2). Each heap H_i is protected by a lock $Lock_i$. Each processor locks the entire heap H_i whenever it makes an operation on H_i.

In the sequential B&B algorithm, we use DFS strictly so H_i is used only if the heaps at higher level (higher on the tree, lower level number) are all empty. In the parallel version, to allow multiple processors shared access to the search space, a processor uses H_i if all the heaps at higher levels are empty or locked by other processors.

The shared-memory B&B framework has a simple termination detection. A processor can terminate its execution when it detects that all the heaps are unlocked and empty: there are no more active nodes except for those being decomposed by other

processors. This is correct, but it could be inefficient, since still-active processors could produce more parallel work for the prematurely-halted processors. If the machine supports it, instead of terminating, a processor can declare itself idle (e.g. by setting a unique bit) and go to sleep. An active processor can then wake it up if there's sufficient new work in the system. The last active processor terminate all sleeping processors and then terminates itself.

1.4 Impact of Parallelization

There are a variety of software packages to reconstruct sequence-based phylogeny. The most popular phylogeny software suites that contain parsimony methods are PAUP* by Swofford [20], PHYLIP by Felsenstein [21], and TNT and NONA by Goloboff [22,23]. We have developed a freely-available shared-memory code for computing MP, that is part of our software suite, GRAPPA (Genome Rearrangement Analysis through Parsimony and other Phylogenetic Algorithms) [2]. GRAPPA was designed to re-implement, extend, and especially speed up the breakpoint analysis (BPAnalysis) method of Sankoff and Blanchette [24]. Breakpoint analysis is another form of parsimony-based phylogeny where species are represented by ordered sets of genes and distances is measured relative to differences in orderings. It is also solved by branch and bound. One feature of our MP software is that it does not constrain the character states of the input and can use real molecular data and also characters reduced from gene-order data such as Maximum Parsimony on Binary Encodings (MPBE) [25].

The University of New Mexico operates *Los Lobos*, the NSF/Alliance 512-processor Linux supercluster. This platform is a cluster of 256 IBM Netfinity 4500R nodes, each with dual 733 MHz Intel Xeon Pentium processors and 1 GB RAM, interconnected by Myrinet switches. We ran *GRAPPA* on *Los Lobos* and obtained a 512-fold speed-up (linear speedup with respect to the number of processors): a complete breakpoint analysis (with the more demanding inversion distance used in lieu of breakpoint distance) for the 13 genomes in the Campanulaceae data set ran in less than 1.5 hours in an October 2000 run, for a *million-fold* speedup over the original implementation [26,1]. Our latest version features significantly improved bounds and new distance correction methods and, on the same dataset, exhibits a speedup factor of *over one billion*. In each of these cases a factor of 512 speed up came from parallelization. The remaining speed up came from algorithmic improvements and improved implementation.

2. Boosting Phylogenetic Reconstruction Methods Using Recursive-Iterative-DCM3

Reconstructing the Tree of Life, i.e., the evolutionary tree of all species on Earth, poses a highly challenging computational problem. Various software packages such as TNT [27–29], PAUP* [30], and RAxML [31] contain sophisticated search procedures for solving MP (Maximum Parsimony) and ML (Maximum Likelihood) on very large datasets. (Section 3 of this chapter describes aspects of the RAxML method in more detail.) The family of Disk Covering Methods (DCMs) [32–35] was introduced to *boost* the performance of a given base method without making changes to the method itself, i.e. use the same search procedures in the base method, except deploy them in a divide and conquer context. DCMs decompose the input set of species (i.e. alignment) into smaller subproblems, compute subtrees on each subproblem using a given *base method*, merge the subtrees to yield a tree on the full dataset, and refine the resulting supertree to make it binary if necessary. Figure 2 shows the four steps of the DCM2 method which was developed for boosting MP and ML heuristics. Figure 3 illustrates the operation of the Strict Consensus Merger supertree method (SCM) which is used for merging the subtrees computed by the base method. SCM is a fast consensus based method that has shown to be more accurate and faster than the Matrix Representation using Parsimony (MRP) method for supertree reconstruction on DCM subproblems [36]. DCMs have previously been shown to significantly improve upon NJ, the most widely used distance-based

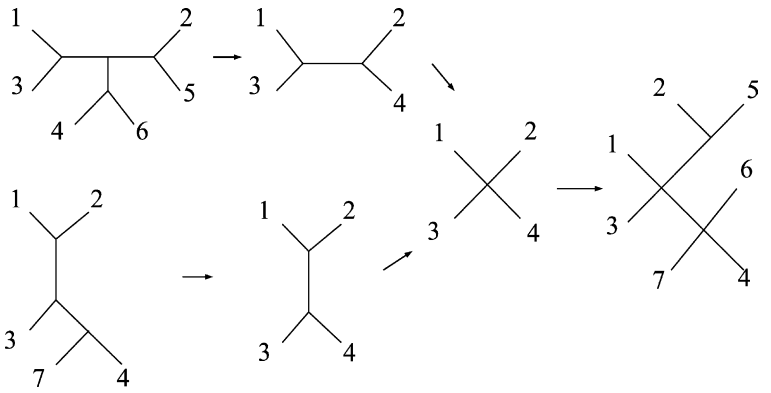

FIG. 3. The Strict Consensus Merger is a consensus-based supertree method that is fast and accurate enough on DCM decompositions. As the figure shows, two subtrees are merged by first computing the set of common taxa and restricting both the input trees to this set. The strict consensus tree, i.e. set of common bipartitions, is computed on the restricted subtrees, and the remaining bipartitions on the two input trees are then attached to the consensus.

method for phylogeny reconstruction. See [37–40] for various studies showing DCM improvements over NJ.

Rec-I-DCM3 is the latest in the family of Disk Covering Methods (DCMs) and was designed to improve the performance of MP and ML heuristics. Rec-I-DCM3 is an iterative technique which uses the DCM3 decomposition [34] for escaping local minima. Previously it was shown that Rec-I-DCM3 improves upon heuristics for solving MP [34,35]. In this study we show that Rec-I-DCM3 combined with RAxML-III finds highly optimal ML trees, particularly on large datasets. Within the current Section we will refer to RAxML-III as RAxML (as opposed to Section 3 where RAxML refers to RAxML-VI).

We first discuss an essential component of Rec-I-DCM3 which is the DCM3 decomposition. We then describe Rec-I-DCM3 in detail and examine its performance in conjunction with RAxML as the base method.

2.1 DCM3 Decomposition

DCM3 is the latest decomposition technique in the family of DCMs. DCM3 was designed as improvement over the previous DCM2 decomposition. As shown previously DCM2 is too slow on large datasets and more importantly, does not always produces subsets that are small enough to give a substantial speedup [35]. The DCM3 decomposition is similar in many ways to the DCM2 technique; the main difference between the two techniques is that DCM2's decomposition is based upon a distance matrix computed on the dataset, while DCM3's decomposition is obtained on the basis of a "guide tree" for the dataset.

We assume we have a tree T on our set S of taxa, and an edge weighting w of T (i.e., $w : E(T) \rightarrow \Re^+$). Based upon this edge-weighted tree, we obtain a decomposition of the leaf set using the following steps. Before describing the decomposition we define the short subtree of an edge.

Short subtrees of edges. Let A, B, C, and D be the four subtrees around e and let a, b, c, and d be the set of leaves closest to e in each of the four subtrees A, B, C, and D respectively (where the distance between nodes u and v is measured as $\sum_{e \in P_{uv}} w(e)$). The set of nodes in $a \cup b \cup c \cup d$ is the "short subtree" around the edge e. We will say that i and j are in a short subtree of T if there is some edge so that i and j are in the short subtree around e. The graph formed by taking the union of all the cliques on short subtrees is the *short subtree graph* and is shown to be triangulated [35].

We begin the decomposition by first constructing the *short subtree graph*, which is the union of cliques formed on "short subtrees" around each edge. Since the short subtree graph G is triangulated, we can find maximal clique separators in polynomial time (these are just cliques in the graph, as proven in [41]), and hence we can find

DCM3 decomposition

- **Input**
  - Set $S = \{s_1, \ldots, s_n\}$ of n aligned biomolecular sequences
  - An edge-weighted phylogenetic *guide tree* T leaf-labeled by S.
- **Output** $A_i, \ldots, A_k$ with $\bigcup_i A_i = S$, and set $X \subset S$ such that $A_i \cap A_j = X$ for all i, j.
- **Algorithm**
  - Compute the *short subtree graph* $G = (V, E)$ where $V = S$ and $E = \{(i, j)\colon i, j \in \text{short subtree of } T\}$.
  - Find a clique separator X in G which minimizes $\max_i |A_i \cup X|$ where $A_1, \ldots, A_k$ are the connected components of G after removing X

FIG. 4. Algorithmic description of the DCM3 decomposition.

(also in polynomial time) a clique separator X that minimizes $\max_i |X \cup C_i|$, where $G - X$ is the union of k components $C_1, C_2, \ldots, C_k$. This is the same decomposition technique used in the DCM2 decomposition, but there the graph is constructed differently, and so the decomposition is different. Figure 4 describes the full DCM3 decomposition algorithm and Fig. 5 shows a toy example of the DCM3 decomposition on a eight taxon phylogeny. We now analyze the running time to compute the DCM3 decomposition.

Theorem 1. *Computing a DCM3 decomposition takes* $O(n^3)$ *time in the worst case, where n is the number of sequences in the input.*

Proof. In the worst case, the input tree can be ultrametric which causes each short subtree to be of size $O(n)$. Thus, for each internal edge ($O(n)$ time) we create a clique for each short subtree ($O(n^2)$ worst case time); the total time taken is $O(n^3)$. The optimal separator and the associated connected components are found by computing a depth-first search ($O(n^2)$ worst case time) for each of the $O(n)$ clique separators; total time taken is $O(n^3)$. Thus, the worst case time is $O(n^3)$. □

Although finding the optimal separator takes $O(n^3)$ time, in practice it takes much longer than computing the short subtree. Rather than explicitly seeking a clique separator X in G which minimizes the size of the largest subproblem, we apply a simple heuristic to get a decomposition, which in practice turns out to be a good decomposition. We explain this heuristic below.

Approximate centroid-edge decomposition. It has been observed on several real datasets that the optimal separator is usually the one associated with the short sub-

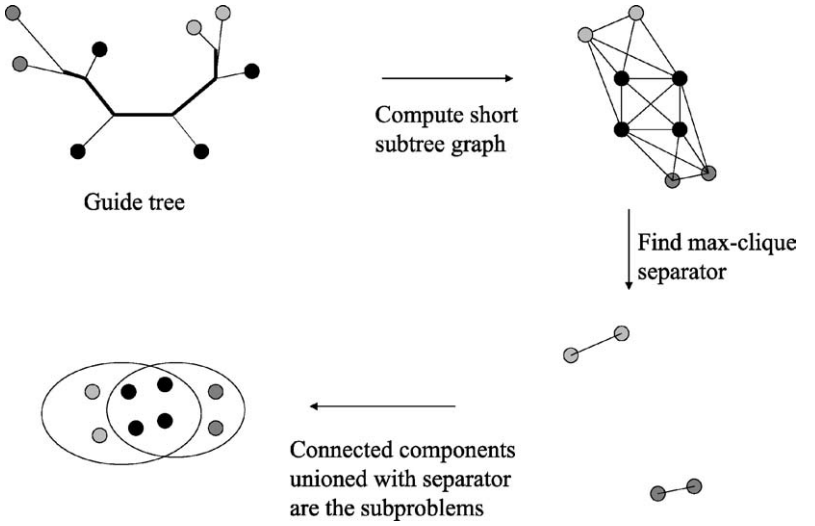

FIG. 5. DCM3 decomposition shown on an eight taxon phylogeny.

tree of the centroid edge [35], i.e., the edge such that when removed produces subtrees of equal size (in number of leaves). This observation allows us to bypass the computation time associated with dealing with short subtrees. We can compute an *approximated centroid edge* decomposition by finding the centroid edge e and setting the separator to be the closest leaves in each of the subtrees around e. The remaining leaves in each of the subtrees around e (unioned with the separator) would then form the DCM3 subproblems (see Fig. 6). This takes linear time if we use a depth-first search. In the rest of this chapter we will use the approximate centroid edge decomposition when we refer to a DCM3 decomposition.

2.2 Recursive-Iterative-DCM3 (Rec-I-DCM3)

Recursive-Iterative-DCM3 is the state of the art in DCMs for solving NP-hard optimization problems for phylogeny reconstruction. It is an iterative procedures which applies an existing base method to both DCM3 subsets and the complete dataset. Rec-I-DCM3 can also be viewed as an iterated local search technique [42] which uses a DCM3 decomposition to escape local minima. Figure 7 provides a full description of the Rec-I-DCM3 algorithm.

The Recursive-DCM3 routine performs the work of dividing the dataset into smaller subsets, solving the subproblems (using the base method), and then merg-

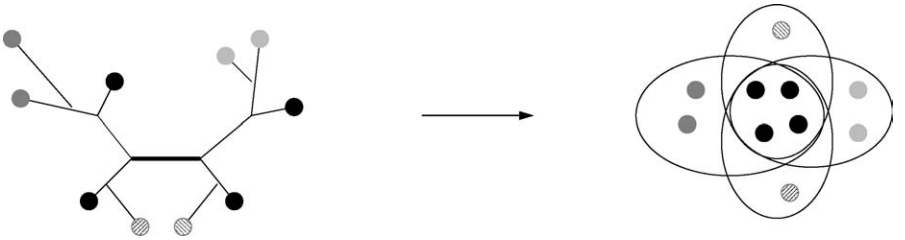

FIG. 6. The faster approximate DCM3 centroid decomposition can be done in O(n) time. Both, finding the centroid edge and computing the subsets, can be done in O(n) using a depth first search (n is the number of leaves).

Recursive-Iterative-DCM3

- **Input**
  - Input alignment S, #iterations n, base heuristic b, global search method g, starting tree T, maximum subproblem size m
- **Output** Phylogenetic tree leaf-labeled by S.
- **Algorithm** For each iteration do
  - Set $T' =$ Recursive-DCM3(S, m, b, T).
  - Apply the global search method g starting from T' until we reach a local optimum. This step can be skipped; however, it usually leads to more optimal trees even with a superficial search.
  - Let T'' be the resulting local optimum from the previous step. Set $T = T''$.

FIG. 7. Algorithm for Recursive-Iterative-DCM3.

ing the subtrees into the full tree. Recursive-DCM3 is a simple modification of the original DCM3. It is obtained by recursively applying the DCM3 decomposition to DCM3 subsets in order to yield smaller subproblems. The sizes of individual subproblems vary significantly and the inference time per subproblem is not known a priori and difficult to estimate. This can affect performance if the subproblems are solved in parallel [43]. The global search method further improves the accuracy of the Recursive-DCM3 tree and can also find optimal global configurations that were not found by Recursive-DCM3, which only operates on smaller—local—subsets. However, previous studies [34,35] and results presented in this one show that even a superficial search can yield good results.

2.3 Performance of Rec-I-DCM3 for Solving ML

Rec-I-DCM3 has previously shown to boost TNT (currently the fastest software package for solving MP) with the default settings of TNT. In this chapter we set out to determine if Rec-I-DCM3 can improve upon the standard hill-climbing algorithms of RAxML (as implemented in RAxML-III). We study the performance of RAxML and Rec-I-DCM3(RAxML) on several real datasets described below.

2.3.1 Real Datasets

We collected 20 real datasets of different sizes, sequence lengths, and evolutionary rates from various researchers. All the alignments were prepared by the authors of the datasets. It is important to use a reliable alignment when computing phylogenies. Therefore, we minimize the use of machine alignments, i.e., those created solely by a computer program with no human intervention. Below we list the size of each alignment along with its sequence length and source.

1. 101 RNA, 1858 bp [44], obtained from Alexandros Stamatakis.
2. 150 RNA, 1269 bp [44], obtained from Alexandros Stamatakis.
3. 150 ssu rRNA, 3188 bp [45], obtained from Alexandros Stamatakis.
4. 193 ssu rRNA [46], obtained from Alexandros Stamatakis.
5. 200 ssu rRNA, 3270 bp [45], obtained from Alexandros Stamatakis.
6. 218 ssu rRNA, 4182 bp [47], obtained from Alexandros Stamatakis.
7. 250 ssu rRNA [45], obtained from Alexandros Stamatakis.
8. 439 Eukaryotic rDNA, 2461 bp [48], obtained from Pablo Goloboff.
9. 476 Metazoan DNA, 1008 bp, created by Doug Ernisse but unpublished, obtained from Pablo Goloboff with omitted taxon names.
10. 500 rbcL DNA, 1398 bp [49].
11. 567 three-gene (rbcL, atpB, and 18s) DNA, 2153 bp [50].
12. 854 rbcL DNA, 937 bp, created by H. Ochoterena but unpublished, obtained from Pablo Goloboff with omitted taxon names.
13. 921 Avian Cytochrome *b* DNA, 713 bp [51].
14. 1000 ssu rRNA, 5547 bp [45], obtained from Alexandros Stamatakis.
15. 1663 ssu rRNA, 1577 bp [45], obtained from Alexandros Stamatakis.
16. 2025 ssu rRNA, 1517 bp [45], obtained from Alexandros Stamatakis.
17. 2415 mammalian DNA, created by Olaf Bininda-Emonds but unpublished, obtained from Alexandros Stamatakis.
18. 6722 three-domain (Eukarayote, Archea, and Fungi) rRNA, 1122 bp, created and obtained by Robin Gutell.
19. 7769 three-domain (Eukaryote, Archea, and Fungi) + two organelle (mitochondria and chloroplast) rRNA, 851 bp, created and obtained by Robin Gutell.

20. 8780 ssu rRNA, 1217 bp [45], obtained from Alexandros Stamatakis.

2.3.2 Parameters for RAxML and Rec-I-DCM3

2.3.2.1 RAxML.

We use default settings of RAxML on each dataset. By default RAxML performs a standard hill-climbing search for ML trees but begins with an estimate of the MP tree (constructed using heuristics implemented in Phylip [52]). We use the HKY85 model [53] throughout the study whenever we run RAxML (even as the base and global methods for Rec-I-DCM3). For more details on RAxML we refer the reader to Section 3 of this chapter where RAxML is thoroughly described.

2.3.2.2 Rec-I-DCM3.

We use RAxML with its default settings for the base method. However, when applying RAxML on a DCM3 subproblem, we use the guide-tree restricted to the subproblem taxa as the starting tree for the search (as opposed to the default randomized greedy MP tree). This way the RAxML search on the subset can take advantage of the structure stored in the guide-tree through previous Rec-I-DCM3 iterations. We use the *fast* RAxML search for the global search phase of Rec-I-DCM3. This terminates much quicker than the standard (and more through) hill-climbing search (which is also the default one). We can expect better performance in terms of ML scores if the standard RAxML search was used as the Rec-I-DCM3 global search; however, that would increase the overall running time. The initial guide-tree for Rec-I-DCM3 is the same starting tree used by RAxML and the Rec-I-DCM3 search was performed for the same amount of time as the unboosted RAxML. The maximum subproblem size of Rec-I-DCM3 was selected as follows:

- 50% for datasets below 1000 sequences;
- 25% for datasets between 1000 and 5000 sequences (including 1000);
- 12.5% for datasets above 5000 sequences (including 5000).

These subproblem sizes may not yield optimal results for Rec-I-DCM3(RAxML). We selected these based upon performance of Rec-I-DCM3(TNT) [34,35] for boosting MP heuristics.

2.3.3 Experimental Design

On each dataset we ran 5 trials of RAxML since each run starts from a randomized greedy MP tree (see [35] and Section 3 for a description of this heuristic). We ran 5 trials of Rec-I-DCM3(RAxML) and report the average best score found by each

method on each dataset. We also report the difference in likelihood scores both in absolute numbers and percentages.

2.3.4 Results

Table I summarizes the results on all the real datasets. The − log likelihood improvement is the average RAxML score subtracted from the Rec-I-DCM3(RAxML) score. This is also shown as a percentage by dividing the improvement by the RAxML average score.

Rec-I-DCM3(RAxML) improves RAxML on 15 of the 20 datasets studied here. On datasets below and including 500 taxa Rec-I-DCM3(RAxML) improves upon

TABLE I
THE DIFFERENCE BETWEEN THE REC-I-DCM3(RAXML) AND RAXML − log LIKELIHOOD SCORE AND ALSO PRESENTATION OF IT AS A PERCENTAGE OF THE RAXML − log LIKELIHOOD SCORE

Dataset size	Improvement as %	− LH Improvement	Max p-distance
101	−0.004	−2.7	0.45
150 (SC)	0.007	3.2	0.43
150 (ARB)	0	0.3	0.54
193	0.06	38.6	0.78
200	−0.006	−6.5	0.54
218	0.014	21	0.42
250	0.014	19	0.55
439	0	0.1	0.65
476	−0.004	−4	0.89
500	0.011	11	0.18
567	0.006	13.9	0.33
854	0.03	42	0.32
921	0.06	109.6	0.39
1000	0.031	123	0.55
1663	−0.004	−11.7	0.48
2025	−0.002	−6	0.56
2415	0.004	23	0.48
6722	1.251	6877	1
7769	2.338	13290	1
8780	0.03	270	0.55

The negative percentages show where RAxML performed better than Rec-I-DCM(RAxML). These percentages are small (at least −0.006%) and show that Rec-I-DCM3(RAxML) performs almost as well as the unboosted RAxML when it fails to provide a better score. We also list the maximum p-distance of each dataset to indicate its divergence. On most of the divergent datasets Rec-I-DCM3(RAxML) improves over RAxML by a larger percentage as opposed to the more conserved ones.

RAxML in 7 out of 10 datasets. The maximum improvement is 0.06% which is on the most divergent dataset of 193 taxa. On datasets above 500 taxa Rec-I-DCM3(RAxML) improves RAxML in 8 out of 10 datasets with the improvement generally more pronounced. On 6 datasets the improvement is above 0.02% and above 1% on the 6722 and 7769 taxon datasets—these two datasets are also highly divergent (as indicated by their maximum pairwise p-distances) and can be considered as very challenging to solve. Interestingly, Rec-I-DCM3(RAxML) does not improve RAxML on the 1663 and 2025 taxon datasets despite their large sizes and moderate maximum p-distances. As indicated by the small percentage values (see Table I) Rec-I-DCM3(RAxML) is almost as good as RAxML on these datasets. It is possible there are certain characteristics of these datasets that make them unsuitable for boosting or for divide-and-conquer methods. We intend to explore this in more detail in subsequent studies.

Figures 8 through 11 show the performance of Rec-I-DCM3(RAxML) and RAxML as a function of time on all the datasets. Each data point in the curve is the average of five runs. Variances are omitted from the figures for the purpose of visual clarity and are in general small. The first time point shown on each graph is the time when the Rec-I-DCM3(RAxML) outputs its first tree, i.e., the tree at the end of the first iteration. This tree is always much better in score than the initial guide-tree. Of the 15 datasets where Rec-I-DCM3(RAxML) has a better score than RAxML at the end of the searches, Rec-I-DCM3(RAxML) improves RAxML at every time point on 11 of them. On the remaining 4 RAxML is doing better initially; however, at the end of the search Rec-I-DCM3(RAxML) comes out with a better score.

2.3.5 Conclusions

Our results indicate that Rec-I-DCM3 can improve RAxML on a wide sample of DNA and RNA datasets. The improvement is larger and more frequent on large datasets as opposed to smaller ones. This is consistent with Rec-I-DCM3 results for boosting MP heuristics [35].

The results presented here are using the algorithms of RAxML-III. It remains to be see how the performance of Rec-I-DCM3(RAxML) will be affected if different (and better) ML hill-climbing algorithms are used (such as those implemented in RAxML-VI). We recommend the user to experiment with different subset sizes (such as one-half, one-quarter, and one-eighth the full dataset size) and both, a standard (and thorough) hill-climbing as well as a superficial one for the global search phase of Rec-I-DCM3. Preliminary results (not shown here) show similar improvements of RAxML-VI using Rec-I-DCM3(RAxML-VI) on some of the datasets used in this study.

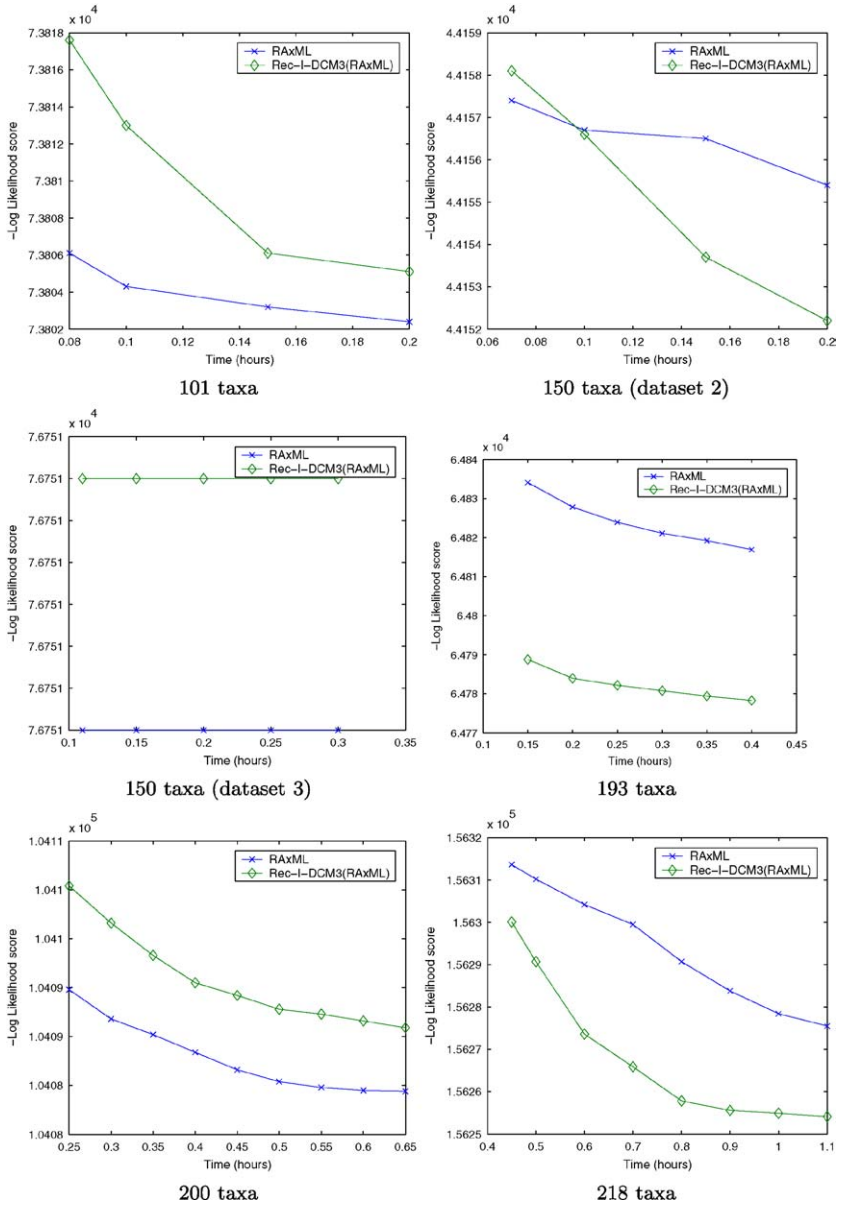

FIG. 8. Rec-I-DCM3(RAxML) improves RAxML on the 150 (dataset 2), 193, and 218 taxon datasets shown here.

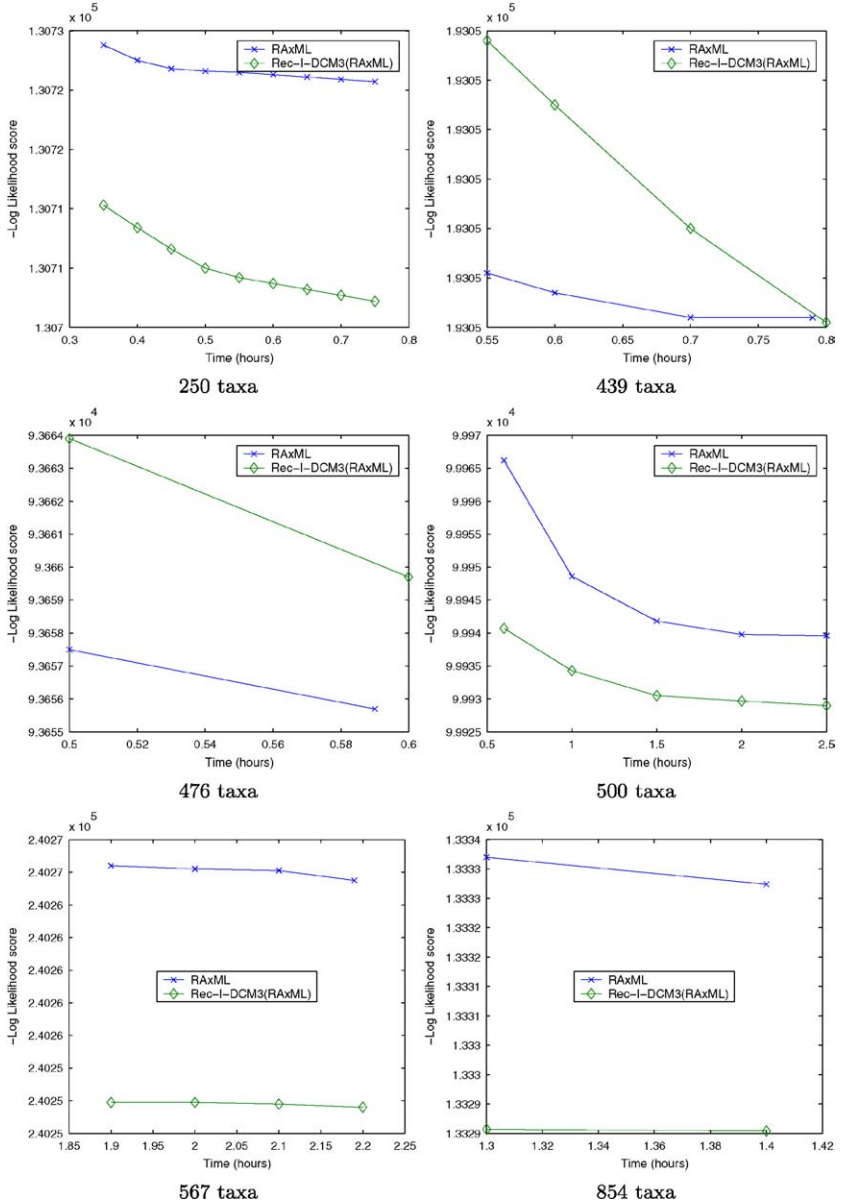

FIG. 9. As the dataset sizes get larger Rec-I-DCM3(RAxML) improves RAxML on more datasets. Here we see improvements on the 250, 500, 567, and 854 taxon datasets.

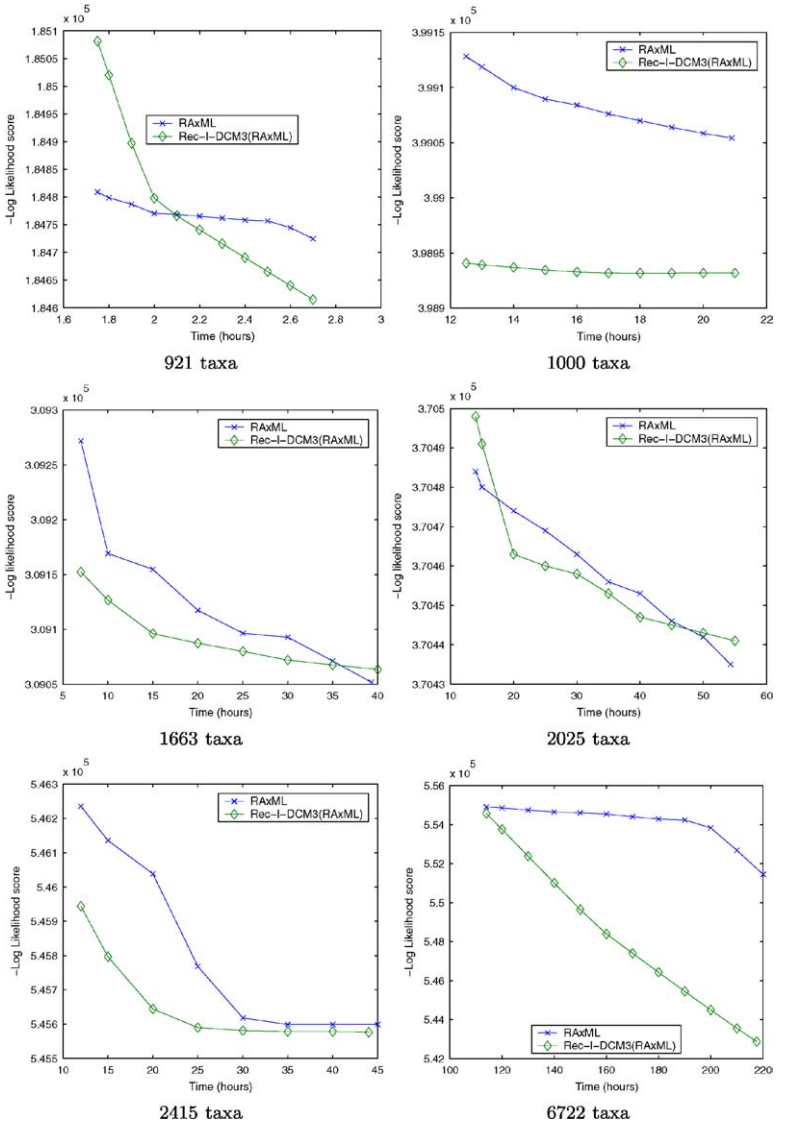

FIG. 10. Rec-I-DCM3(RAxML) improves RAxML on all the datasets shown here except for the 1663 and 2025 taxon ones. There we see that Rec-I-DCM3(RAxML) improves RAxML in the earlier part of the search but not towards the very end. It is possible these datasets possess certain properties which make it hard for booster methods like Rec-I-DCM3. On the 6722 taxon dataset we see a very large improvement of with Rec-I-DCM3(RAxML) (of over 1%—see Table I).

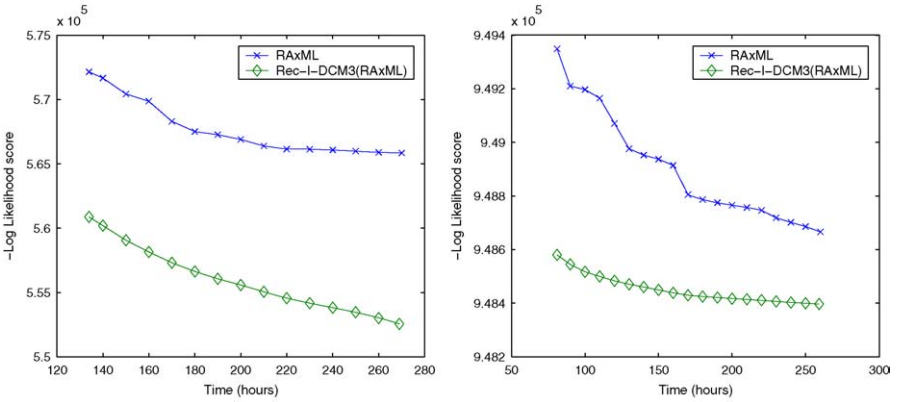

FIG. 11. Rec-I-DCM3(RAxML) improves RAxML on the two largest datasets. On the 7769 taxon dataset the improvement in score is 2.34% which is the largest over all the datasets examined here.

3. New Technical Challenges for ML-Based Phylogeny Reconstruction

The current Section intends to cover the relatively *new* phenomenon of technical problems which arise for the inference of large phylogenies—containing more than 1000 organisms—with the popular Maximum Likelihood (ML) method [54].

The tremendous accumulation of sequence data over recent years coupled with the significant progress in search (optimization) algorithms for ML and the increasing use of parallel computers allow for inference of huge phylogenies within less than 24 hours. Therefore, large-scale phylogenetic analyses with ML are becoming more common recently [55].

The computation of *the* ML tree has recently been demonstrated to be NP-complete [56]. The problem of finding the optimal ML tree is particularly difficult due to the immense amount of alternative tree topologies which have to be evaluated *and* the high computational cost—in terms of floating point operations—of each tree evaluation per se. To date, the main focus of researchers has been on improving the search algorithms (RAxML [57], PHYML [58], GAML [59], IQPNNI [46], MetaPIGA [60], Treefinder [61]) and on accelerating the likelihood function via algorithmic means by detecting and re-using previously computed values [62,63].

Due to the algorithmic progress which has been achieved there exists a noticeable number of programs which are now able to infer a sufficiently accurate 1000-taxon tree within less than 24 hours on a single PC processor.

However, due to the increasing size of the data and the complexity of the more elaborate models of nucleotide substitution, a new category of technical problems arises. Those problems mainly concern cache efficiency, memory shortage, memory organization, efficient implementation of the likelihood function (including manual loop unrolling and re-ordering of instructions), as well as the use of efficient data-structures.

The main focus of this section is to describe those problems and to present some recent technical solutions. Initially, Section 3.1 briefly summarizes the basic mathematics of ML in order to provide a basic understanding of the compute-intensive likelihood function. The following Section 3.2 covers some of the most recent and most efficient state-of-the-art ML phylogeny programs and shows that performance of most programs is currently *limited by memory efficiency and consumption*. In Section 3.3 the data-structures, memory organization, and implementation details of RAxML are described. RAxML has inherited an excellent technical implementation from fastDNAml which has unfortunately never been properly documented. Finally, Section 3.4 covers applications of HPC techniques and architectures to ML-based phylogenetic inference.

3.1 Introduction to Maximum Likelihood

This section does not provide a detailed introduction to ML for phylogenetic trees. The goal is to offer a notion of the complexity and amount of arithmetic operations required to compute the ML score for one *single* tree topology. The seminal paper by Felsenstein [54] which introduces the application of ML to phylogenetic trees and the comprehensive and readable chapter by Swofford et al. [64] provide detailed descriptions of the mathematical as well as computational background.

To calculate the likelihood of a *given* tree topology with *fixed* branch lengths a probabilistic model of nucleotide substitution $P_{ij}(t)$ is required which allows for computing the probability P that a nucleotide i mutates to another nucleotide j within time t (branch length). The model for DNA data must therefore provide substitution transitions:

```
A|C|G|T -> A|C|G|T
```

In order to significantly reduce the mathematical complexity of the overall method the model of nucleotide substitution must be time-reversible [54], i.e. the evolutionary process has to be identic if followed forward or backward in time. Essentially, this means that the maximum number of possible transition types in the General Time Reversible model of nucleotide substitution (GTR [65,66]) is reduced to 6 due to required symmetries. The less general time-reversible models

of nucleotide substitution such as the Jukes–Cantor (JC69 [67]) or Hasegawa–Kishino–Yano (HKY85 [68]) model can be derived from GTR by further restriction of possible transition types. It is important to note, that there exists a trade-off between speed and quality among substitution models. The simple JC69 model which only has one single transition type requires significantly less floating point operations to compute $P_{ij}(t)$ than GTR which is the most complex and accurate one.

Thus, model selection has a significant impact on inference times, and therefore—whenever possible—the simpler model should be used for large datasets, e.g. HKY85 instead of GTR. The applicability of a less complex model to a *specific* alignment can be determined by application of likelihood ratio tests. Thus, if the likelihood obtained for a fixed tree topology with HKY85 is not significantly worse than the GTR-based likelihood value, HKY85 should be used. Programs such as Modeltest [69] can be applied to determine the appropriate model of evolution for a specific dataset.

Another very important and rarely discussed modeling issue concerns the way rate heterogeneity among sites (alignment columns) is accommodated in nucleotide substitution models (see discussion on page 153). There exist two competing models which differ significantly in terms of amount of floating point operations and memory consumption.

Given the model of nucleotide substitution and a tree topology with branch lengths where the data (the individual sequences of the multiple alignment) is located at the tips, one can proceed with the computation of the likelihood score for that tree. In order to compute the likelihood a *virtual root* (*vr*) has to be placed into an *arbitrary* branch of the tree in order to calculate/update the individual entries of each *likelihood vector* (also often called partial likelihood) with length m (alignment length) in the tree bottom-up, i.e. starting at the tips and moving towards *vr*. If the model of nucleotide substitution is time-reversible the likelihood of the tree is identic irrespectively of where *vr* is placed. After having updated all likelihood vectors the vectors to the right and left of *vr* can be used to compute the overall likelihood of the tree.

Note that, the number n (where n is the number of taxa) and length of likelihood vectors m (where m is the number of distinct patterns/columns in the alignment) dominate the memory consumption of typical ML implementations which is thus $O(n * m)$. Section 3.3 describes how the likelihood vector structures can efficiently be implemented to consume only $\Theta(n * m)$ memory.

The process of rooting the tree at *vr* and updating the likelihood vectors is outlined in Fig. 12 for a 4-taxon tree.

To understand how the individual likelihood vectors are updated consider a subtree rooted at node `p` with immediate descendants `r` and `q` and likelihood vectors `l_p`, and `l_q`, `l_r` respectively. When the likelihood vectors `l_q` and `l_r` have

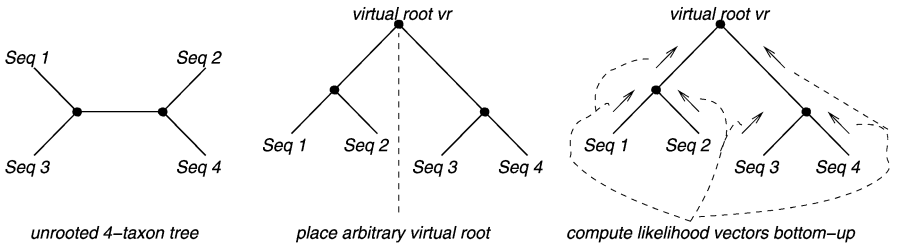

FIG. 12. Computation of the likelihood vectors of 4-taxon tree.

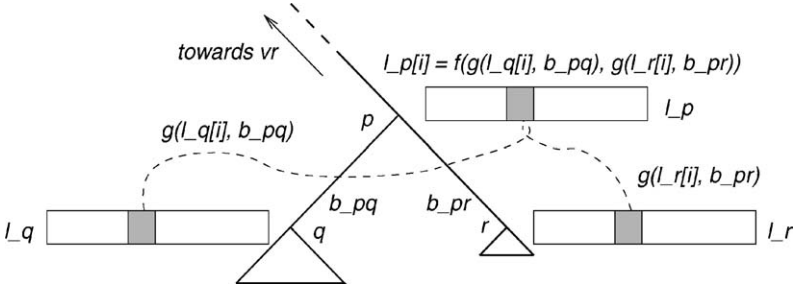

FIG. 13. Updating the likelihood vector of node p at position i.

been computed the entries of `l_p` can be calculated—in an extremely simplified manner—as outlined by the pseudo-code below and in Fig. 13:

```
for(i = 0; i < m; i++)
l_p[i] = f(g(l_q[i], b_pq), g(l_r[i], b_pr));
```

where `f()` is a simple function, i.e. requires just a few FLOPs, to combine the values of `g(l_q[i], b_pq)` and `g(l_r[i], b_pr)`. The `g()` function however is more complex and computationally intensive since it calculates $P_{ij}(t)$. The parameter t corresponds to the branch lengths `b_pq` and `b_pr` respectively. Note, that the `for`-loop can easily be parallelized on a fine-grained level since entries `l_p[i]` and `l_p[i + 1]` can be computed independently (see Section 3.4).

Up to this point it has been described how to compute the likelihood of a tree given some arbitrary branch lengths. In order to obtain the *maximum* likelihood value for a given tree topology the length of *all* branches in the tree has to be optimized. Since the likelihood of the tree is not altered by distinct rootings of the tree the virtual root can be subsequently placed into all branches of the tree. Each branch can then be individually optimized to improve the likelihood value of the entire tree. In general—depending on the implementation—this process is continued until no further branch

length alteration yields an improved likelihood score. Branch length optimization can be regarded as maximization of a one-parameter function $lh(t)$ where lh is the phylogenetic likelihood function and t the branch length.

Some of the most commonly used optimization methods are the Newton–Raphson method in fastDNAml [70] or Brent's rule in PHYML [58].

Typically, the two basic operations: computation of the likelihood value and optimization of the branch lengths, require $\approx 90\%$ of the complete execution time of every ML program. For example 92.72% of total execution time for a typical dataset with 150 sequences in PHYML and 92.89% for the same dataset in RAxML-VI. Thus, an acceleration of these functions at a technical level by optimization of the source code and the memory access behavior, or at an algorithmic level by re-use of previously computed values is very important.

A technically extremely efficient implementation of the likelihood function has been coded in fastDNAml. The Subtree Equality Vector (SEV) method [63] represents an algorithmic optimization of the likelihood function which exploits alignment pattern equalities to avoid a substantial amount of re-computations of the expensive `g()` function. An analogous approach to accelerate the likelihood function has been proposed in [62].

As already mentioned another important issue within the HPC context is the mathematical accommodation of rate variation (also called rate heterogeneity) among sites in nucleotide substitution methods, since sites (alignment columns) usually do not evolve at the same speed. It has been demonstrated, e.g. in [71], that ML inference under the assumption of rate homogeneity can lead to erroneous results if rates vary among sites.

Rate heterogeneity among sites can easily be accommodated by incorporating an additional per-site (per-alignment-column) rate vector `r[]` of length m into function `g()`.

The pseudocode for updating the likelihood vectors with rate categories is indicated below:

```
for(i = 0; i < m; i++)
l_p[i] = f(g(l_q[i], b_pq, r[i]),
           g(l_r[i], b_pr, r[i]));
```

Often, such an assignment of individual rates to sites corresponds to some functional classification of sites and can be performed based on an a priori analysis of the data. G. Olsen has developed a program called DNArates [72] which performs an ML estimate of the individual per site substitution rates for a given input tree. A similar technique is used in RAxML and the model is called e.g. GTR + CAT to distinguish it from GTR + Γ (see below), when the GTR model of nucleotide substitution is used. However, the use of individual per-site rate categories might lead

to over-fitting the data. This effect can be alleviated by using rate categories instead of individual per-site rates, e.g. for an alignment with a length of 1000 base pairs only $c = 25$ or $c = 50$ distinct rate categories are used. To this end an integer vector `category[]` of length m is used which assigns an individual rate category `cat` to each alignment column, where $1 \leqslant$ `cat` $\leqslant c$. The vector `rate[]` of length c contains the rates. This model will henceforth be called CAT model of rate heterogeneity. The abstract structure of a typical `for`-loop to compute the likelihood under CAT is outlined below:

```
 for(i = 0; i < m; i++)
{
 cat = category[i];
 r = rate[cat];
 l_p[i] = f(g(l_q[i], b_pq, r), g(l_r[i], b_pr, r));
}
```

However, little has been published on how to optimize per-site evolutionary rates and how to reasonably categorize per-site evolutionary rates. A notable exception, dealing with per-site rate optimization, is a relatively recent paper by Meyer et al. [73]. The current version of RAxML is one of the few ML programs which implements the CAT model.

A computationally more intensive and thus less desirable form of dealing with heterogeneous rates, due to the fact that significantly more memory *and* floating point operations are required (typically factor 4), consists in using either discrete or continuous stochastic models for the rate distribution at each site. In this case every site has a certain probability of evolving at any rate contained in a given probability distribution. Thus, for a discretized distribution with a number ρ of discrete rates, ρ distinct likelihood vector entries have to be computed *per* site `i`. In the continuous case likelihoods must be integrated over the entire probability distribution.

The most commonly used distributions are the continuous [74] and discrete [71] Γ distributions. Typically, a discrete Γ distribution with $\rho = 4$ points/rates is used since this represents an acceptable trade-off between inference time, memory consumption, and accuracy. Given the *four* individual rates from the discrete Γ distribution `r_0,...,r_3` *now four* individual likelihood entries `l_p[i].g_0,...,l_p[i].g_3` *per* site `i` have to be updated as indicated below:

```
for(i = 0; i < m; i++)
{
l_p[i].g_0 = f(g(l_q[i], b_pq, r_0),
               g(l_r[i], b_pr, r_0));
```

```
  l_p[i].g_1 = f(g(l_q[i], b_pq, r_1),
                 g(l_r[i], b_pr, r_1));
  l_p[i].g_2 = f(g(l_q[i], b_pq, r_2),
                 g(l_r[i], b_pr, r_2));
  l_p[i].g_3 = f(g(l_q[i], b_pq, r_3),
                 g(l_r[i], b_pr, r_3));
  }
```

Usually, Biologists have to account for rate heterogeneity in their analyses due to the properties of real world data and in order to obtain *publishable* results.

From an HPC point of view it is evident that the CAT model should be preferred over the Γ model due to the significantly lower memory consumption and amount of floating point operations which result in faster inference times. However, little is known about the correlation between the CAT and the Γ model, despite the fact that they are intended to model the same phenomenon. A recent experimental study [75] with RAxML on 19 real-world DNA data alignments comprising 73 up to 1663 taxa indicate that CAT is on average over 5 times faster than Γ and—surprisingly enough—also yields trees with even slightly better final Γ likelihood values (factor 1.000014 for 50 rate categories, and factor 1.000037 for 25 rate categories). Similar experimental results have been obtained by Derrick Zwickl on different datasets. Citing from [76], p. 62: "*In practice, performing inferences using the* GTR + CAT *model in RAxML has proven to be an excellent method for obtaining topologies that score well under the* GTR + Γ *model*."

The large speedup of CAT over Γ which exceeds factor 4 is due to increased cache efficiency, since CAT only uses *approximately one quarter* of the memory and the floating point operations required for Γ. In fact, the utilization of Γ lead to an average increase of L2 cache misses by factor 7.46 and factor 7.41 for the L3 cache respectively. Thus, given the computational advantages of CAT over Γ, more effort needs to be invested into the design of a more solid mathematical framework for CAT. The current implementation and categorization algorithm in RAxML has been derived from empirical observations [75]. In addition, final likelihood values obtained under the CAT approximation are numerically instable at present such that the likelihood of final trees needs to be re-computed under Γ in order to compare alternative trees based on their likelihood values. The recently released RAxML manual (www.ics.forth.gr/~stamatak (software frame)) describes this in more detail.

In order to underline the efficiency of the GTR+CAT approximation over GTR+Γ Fig. 14 depicts the GTR + Γ Log Likelihood development over time (seconds) on the same starting tree. This alignment of 8864 Bacteria is currently analyzed with RAxML in cooperation with the Pace Lab at the University of Colorado at Boulder.

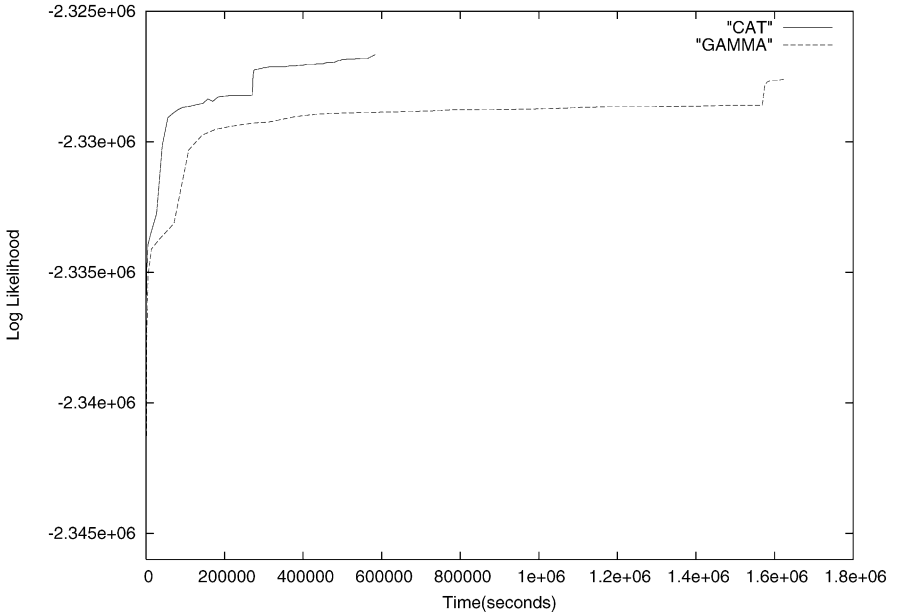

FIG. 14. RAxML Gamma log likelihood development over time for inferences under GTR + CAT and GTR + Γ on an alignment of 8864 bacteria.

3.2 State-of-the-Art Programs

The current Section lists and discusses some of the most popular and widely used sequential and parallel ML programs for phylogenetic inference.

3.2.1 Hill-Climbing Algorithms

In 2003 Guidon and Gascuel published an interesting paper about their very fast program PHYML [58]. The respective performance analysis includes larger simulated datasets of 100 sequences and two well-studied real data sets containing 218 and 500 sequences. Their experiments show that PHYML is extremely fast on real and simulated data.

However, the current hill-climbing and simulated annealing algorithms of RAxML clearly outperform PHYML on real world data, both in terms of execution time and final tree quality [57]. The requirement to improve accuracy on real data [57] and to replace NNI moves (Nearest Neighbor Interchange) by more exhaustive SPR moves (Subtree Pruning Re-grafting, also called subtree rearrangements) has been recognized by the authors of PHYML. In fact, a very promising refinement/extension of

the *lazy subtree rearrangement* technique from RAxML [57] has very recently been integrated into PHYML [77].

Irrespective of these differences between RAxML and PHYML, the results in [58] show that well-established sequential programs like PAUP* [78], TREE-PUZZLE [79], and fastDNAml [70] are prohibitively slow on datasets containing more than 200 sequences, at least in sequential execution mode.

More recently, Vinh et al. [46] published a program called IQPNNI which yields better trees than PHYML on real world data but is significantly slower. In comparison to RAxML, IQPNNI is both slower and less accurate [80].

3.2.2 Simulated Annealing Approaches

The first application of simulated annealing techniques to ML tree searches was proposed by Salter et al. [81] (the technique has previously been applied to MP phylogenetic tree searches by D. Barker [82]). However, the respective program SSA has not become very popular due to the limited availability of nucleotide substitution models and the focus on the molecular clock model of evolution. Moreover, the program is relatively hard to use and comparatively slow in respect to recent hill-climbing implementations. Despite the fact that Salter et al. where the first to apply simulated annealing to ML-based phylogenetic tree searches there do not exist any published biological results using SSA. However, the recent implementation of a simulated annealing search algorithm in RAxML [80] yielded promising results.

3.2.3 Parallel Phylogeny Programs

Despite the fact that parallel implementations of ML programs are technically very solid in terms of performance and parallelization techniques, they significantly lag behind algorithmic development. This means, that programs are parallelized that mostly do not represent the state-of-the-art algorithms any more. Therefore, they are likely to be out-competed by the most recent sequential algorithms in terms of final tree quality and—more importantly—accumulated CPU time.

For example, the largest tree computed with parallel fastDNAml [83] which is based on the fastDNAml algorithm from 1994 contained 150 taxa. Note, that there also exists a distributed implementation of this code [84].

The same argument holds for a technically very interesting JAVA-based distributed ML program: DPRml [85]. Despite the recent implementation of state-of-the-art search algorithms in DPRml, significant performance penalties are caused by using JAVA both in terms of memory efficiency and speed of numerical calculations. Those language-dependent limitations will become more intense when trees comprising over 417 taxa (currently largest tree with DPRml, Thomas Keane, personal communication) are computed with DPRml.

The technically challenging parallel implementation of TrExML [86,87] (original sequential algorithm published in the year 2000) has been used to compute a tree containing 56 taxa. However, TrExML is probably not suited for computation of very large trees since the main feature of the algorithm consists in a more exhaustive exploitation of search space for medium-sized alignments. Due to this exhaustive search strategy the execution time increases more steeply with the number of taxa than in other programs.

The largest tree computed with the parallel version of TREE-PUZZLE [88] contained 257 taxa due to limitations caused by the data structures used (Heiko Schmidt, personal communication). However, TREE-PUZZLE provides mainly advantages concerning quality-assessment for medium-sized trees. IQPNNI has also recently been parallelized with MPI and shows good speedup values [89].

M.J. Brauer et al. [59] have implemented a parallel genetic tree-search algorithm (parallel GAML) which has been used to compute trees of up to approximately 3000 taxa with the main limitation for the computation of larger trees being memory consumption (Derrick Zwickl, personal communication). However, the new tree search mechanism implemented in the successor of GAML, which is now called GARLI [76] (Genetic Algorithm for Rapid Likelihood Inference, available at http://www.bio.utexas.edu/grad/zwickl/web/garli.html) is equally powerful as the RAxML algorithm (especially on datasets $\leqslant$1000 taxa) but requires higher inference times [76]. However, GARLI is one of the few state-of-the-art programs, that incorporates an outstanding technical implementation and optimization of the likelihood functions.

There also exists a parallel version of Rec-I-DCM3 [34] for ML which is based on RAxML (see Section 2.2 of this chapter). The current implementation faces some scalability limitations due to load imbalance caused by significant differences in the subproblem sizes. In addition, the parallelization of RAxML for global tree optimizations also faces some intrinsic difficulties (see [90] and page 168 in Section 3.4.3).

Finally, there exist the previous parallel and distributed implementations of the RAxML hill-climbing algorithm [91,92].

3.2.4 Conclusion

The above overview of recent algorithmic and technical developments, and the maximum tree sizes calculated so far, underlines the initial statement that a part of the computational problems in phylogenetics tends to become technical. This view is shared in the recent paper by Hordijk and Gascuel on the new search technique implemented in PHYML [77]. In order to enable large-scale inference of huge trees a greater part of research efforts should focus on the technical implementation of the likelihood functions, the allocation and use of likelihood vectors, cache efficiency,

as well as exploitation of hardware such as Symmetrical Multi-Processing (SMPs), Graphics Processing Units (GPUs), and Multi-Core Processors. Thus, the rest of the current section will mainly focus on these rarely documented and discussed technical issues and indicate some potential directions of future research.

3.3 Technical Details: Memory Organization and Data Structures

As already mentioned, the implementation of the likelihood functions in fastDNAml represents perhaps *the* most efficient implementation currently available, both in terms of memory organization and loop optimization. The current version of RAxML has been derived from the fastDNAml source code and extended this efficient implementation.

The current Section reviews some of the—so far undocumented—technical implementation details which will be useful for future ML implementations.

3.3.1 Memory Organization and Efficiency

As outlined in Section 3.1 the amount of memory space required is dominated by the length and number of likelihood vectors. Thus, the memory requirements are of order $O(n * m)$ where n is the number of sequences and m the alignment length. An unrooted phylogenetic tree for an alignment of dimensions $n * m$ has n tips or leaves and $n-2$ inner nodes, such that $2n-2$ vectors of length m would be required to compute the likelihood bottom-up at a given virtual root vr. Note that, the computation of the vectors at the tips of the tree (leaf-vectors) is *significantly less expensive* than the computation of inner vectors. In addition, the values of the leaf-vectors are topology-independent, i.e. it suffices to compute them *once* during the initialization of the program. Unlike most other ML implementations however, in fastDNAml a distinct approach has been chosen: The program trades memory for additional computations, i.e. only 3 (!) likelihood vectors are used to store tip-values. This means that tip likelihood vectors will have to be continuously re-computed on-demand during the entire inference process. On the other hand the memory consumption is reduced to $(n + 1) * m$ in contrast to $(2n - 2) * m$. This represents a memory footprint reduction by almost factor 2. This leads to improved cache-efficiency and the capability to handle larger alignments. Experiments with RAxML using the alternative implementation with n precomputed leaf-vectors on a 1000-taxon alignment have demonstrated that the re-computation of leaf-vector values is in fact more efficient, even with respect to execution times. Due to the growing chasm between CPU and RAM performance and the constantly growing alignment sizes, the above method should be used. The importance and impact of cache efficiency is also underlined

by the significant superlinear speedups achieved by the OpenMP implementation of RAxML (see [93] and Fig. 19).

The idea of trading memory for computation with respect to tip vectors has been further developed in the current release of RAxML-VI for High Performance Computing (RAxML-VI-HPC). This new version does not use or compute any leaf-vectors at all. Instead it uses one global leaf-likelihood vector `globalLeafVector[]` of length 15 which contains the pre-computed likelihood vectors for all 15 possible nucleotide sequence states. Note that, the number of 15 states comes from some intermediate states which are allowed, e.g. apart from `A,C,G,T,-` the letter `R` stands for `A or G` and `Y` for `C or T` etc. When a tip with a nucleotide sequence `sequence[i]` where `i=1,...,m` and alignment length `m` is encountered, the respective leaf-likelihood vector at position `i` of the alignment is obtained by referencing `globalLeafVector[]` via the sequence entry `sequence[i]`, i.e. `likelivector = globalTip[sequence[i]]`. Note that `sequence[]` is a simple array of type `char` which contains the sequences of the alignment. The introduction of this optimization yielded performance improvements of approximately 5–10%. Finally, note that GARLI uses a similar, though more sophisticated implementation of leaf-likelihood vector computations (Derrick Zwickl, personal communication).

With respect to the internal likelihood vectors there also exist two different approaches. In programs such as PHYML or IQPNNI not one but *three* likelihood vectors are allocated to each internal node, i.e. one vector for each direction of the unrooted tree. Thus, PHYML also maintains an unrooted view of the tree with respect to the likelihood vector organization.

If the likelihood needs to be calculated at an arbitrary branch of the tree the required likelihood vectors to the left and right of the virtual root will be immediately available. On the other hand, a very large amount of those vectors will have to be re-computed after a change of the tree topology or branch lengths (see Fig. 15 for an example).

In RAxML and fastDNAml only one inner likelihood vector per internal node, is allocated. This vector is relocated to the one of the three outgoing branches `noderec *next` (see data structure below) of the inner node which points towards the current virtual root. If the likelihood vector `xarray *x` is already located at the correct branch it must not be recomputed. The infrastructure to move likelihood vectors is implemented by a cyclic list of 3 data structures of type `node` (one per outgoing branch `struct noderec *back`) to the likelihood vector data structure. At all times, two of those pointers point to `NULL` whereas one points to the actual address of the likelihood vector (see Fig. 16).

```
typedef struct noderec {
double z; /* branch length value */
```

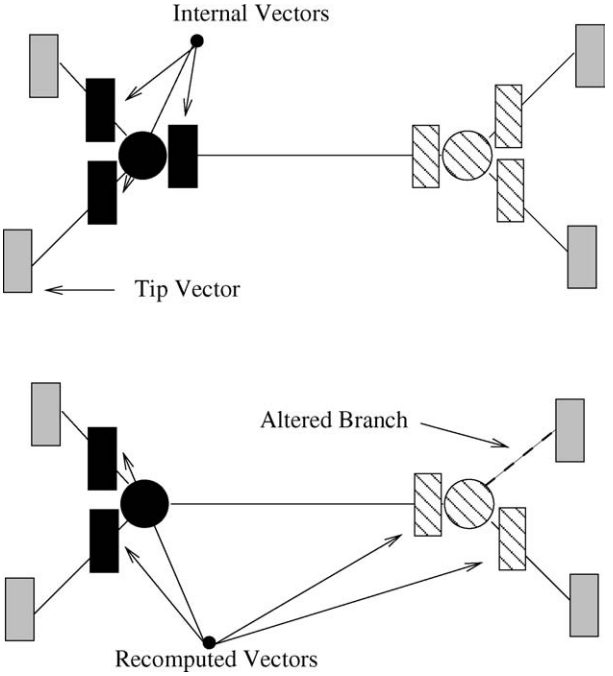

FIG. 15. Likelihood vector update in PHYML.

```
struct noderec *next;
/* pointer to next structure in cyclic list*/
struct noderec *back; /* pointer to neighboring node*/
xarray *x; /* pointer to likelihood vector*/
} node;
```

With respect to the position of the likelihood vectors in the cyclic list of `struct noderec` a tree using this scheme is always rooted. In addition, at each movement of the virtual root, in order to e.g. optimize a branch, a certain amount of vectors must be recomputed. The same holds for changes in tree topology. However, as for the tip vectors, there is a trade-off between additional computations and reduced memory consumption for inner likelihood vectors as well. Moreover, the order by which topological changes are applied to improve the likelihood, can be arranged intelligently, such that only few likelihood vectors need to be updated after each topological change. Currently, there exists no comparative study between those two approaches to memory organization and likelihood calculation. Nonetheless, it

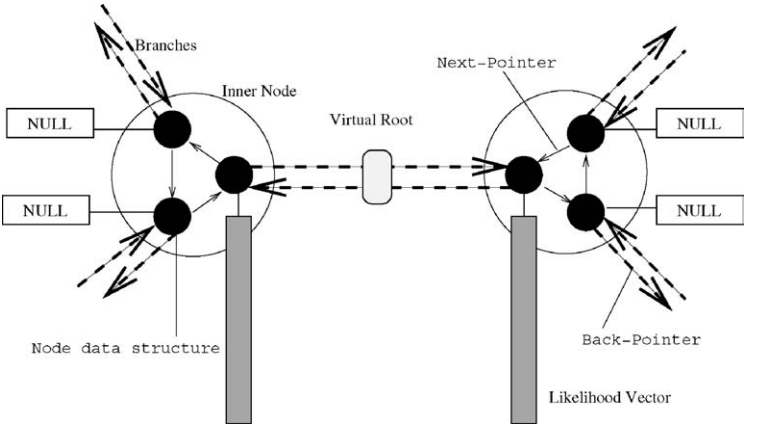

FIG. 16. Likelihood vector organization in RAxML.

would be very useful to compare memory consumption, cache efficiency, and amount of floating point operations for these alternatives under the *same* search algorithm.

It appears however, that the latter approach is more adequate for inference of extremely large trees with ML. Experiments with an alignment of approximately 25,000 protobacteria show that RAxML already requires 1.5 GB of main memory using the GTR + CAT approximation. A very long multi-gene alignment of 2182 mammalian sequences with a length of more than 50,000 base pairs already required 2.5 GB under GTR + CAT and 9 GB under GTR + Γ. To the best of the authors knowledge this alignment represents the largest data matrix which has been analyzed under ML to date. Given that the alternative memory organization requires at least 3 times more memory it is less adequate for inference of huge trees.

One might argue, that the application of a divide-and-conquer approach can solve memory problems since it will only have to handle significantly smaller subtrees and sub-alignments. Due to the algorithmic complexity of the problem however, every divide-and-conquer approach to date also performs *global* optimizations on the complete tree.

The extent of the memory consumption problem becomes even more evident when one considers, that the length m of the 25,000 protobacteria alignment is only 1463 base pairs, i.e. it is relatively short with respect to the large number of sequences. Typically, for a *publishable* biological analysis of such large datasets a significantly greater alignment length would be required [40]. In the final analysis it can be stated that memory organization and consumption are issues of increasing importance in the quest to reconstruct the tree of life which should contain at least 100,000 or 1,000,000 organisms based on the rather more conservative estimates.

The increasing concern about memory consumption is also reflected by the recent changes introduced in the new release of MrBayes [94] (version 3.1.1). Despite the fact that MrBayes performs Bayesian inference of phylogenetic trees the underlying technical problems are the same since the likelihood value of alternative tree topologies needs to be computed and thus likelihood computations consume a very large part of execution time. Therefore, to reduce memory consumption of MrBayes, double-precision arithmetics have been replaced by single-precision operations.

3.3.2 Loop Optimization and Model Implementation

Another aspect of increasing importance in HPC ML program design consists in highly optimized implementations of the likelihood functions. They consume over 90% of total execution time in typical ML implementations, e.g. 92.72% in PHYML and 92.89% in RAxML for a typical dataset of 150 sequences.

Despite the obvious advantages of a generic programming style as used e.g. in PHYML or IQPNNI, each model of sequence evolution such as HKY85 or GTR should be implemented in separate functions. Depending on the selected model RAxML uses function pointers to highly optimized individual functions for each model. This allows for better exploitation of symmetries and simplifications on a per-model basis. As already mentioned the compute-intensive part of the computations is performed by 4–5 `for`-loops (depending on the implementation) over the length of the alignment m. For example the manual optimization and complete unrolling of inner loops for the recently implemented protein substitution models in RAxML yielded more than 50% of performance improvement. This increase in performance could not be achieved by the use of highly sophisticated Intel or PGI compilers alone. In addition, instructions within the for-loops have been re-ordered to better suit pipeline architectures.

Another important technical issue concerns the optimization technique used for branch lengths, which consumes 42.63% of total execution time in RAxML and 58.74% in PHYML. Despite the additional cost required to compute the first and second derivative of the likelihood function, the Newton–Raphson method (RAxML, fastDNAml) should be preferred over Brent's method (PHYML) since Newton–Raphson converges *significantly* faster. Due to this observation Brent has recently been replaced by Newton–Raphson in the new version of IQPNNI [89] (version 3.0). In addition, the latest version of IQPNNI also incorporates the BFGS method [95] for multi-dimensional optimization of model parameters (Bui Quang Minh, personal communication). BFGS is very efficient for parameter-rich models such as GTR + Γ or complex protein models, in comparison to the more common approach of optimizing parameters one-by-one. By deploying BFGS the parameter optimization process for the 6 rate parameters of the GTR model in IQPNNI could be accelerated by factor 3–4 in comparison to Brent (Bui Quang Minh, personal communication). Those

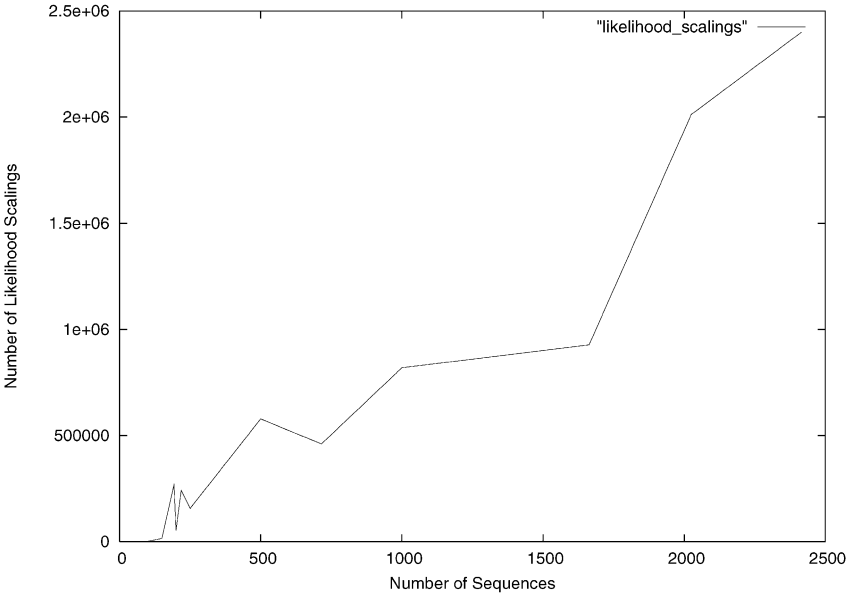

FIG. 17. Number of likelihood scalings.

mathematical improvements lead to a total performance improvement of factor 1.2 up to 1.8 in IQPNNI over the previous version of the program.

A useful discussion of numerical problems and solutions for the inference of large trees can be found in [96]. Another important numerical design decision which concerns the memory-time trade-off is the choice between single (e.g. MrBayes), double (e.g. RAxML, PHYML), and long double (IQPNNI) precision arithmetics for calculating the likelihood. This choice is important since it has an effect on the number of times *very small* likelihood values have to be scaled (scaling events). Typically, the larger the tree, the more *scaling events* are anticipated. This trend is outlined in Fig. 17 where the x-axis indicates the number of sequences in the dataset and the y-axis the number of scaling events in RAxML for the evaluation and parameter-optimization of one single tree topology. When single precision is used those computationally relatively expensive operations have to be performed more frequently. On the other hand double and long double require more memory space. Thus, the choice of double precision appears to represent a reasonable trade-off. A porting of RAxML from `double` to `float` for the purposes of the GPGPU implementation [97] (see Section 3.4) did not yield better results in terms of execution times.

3.4 Parallelization Techniques

Typically, in ML programs there exist three distinct sources of parallelism which are depicted in Fig. 18:

1. *Fine-grained loop-level parallelism* at the `for`-loops of the likelihood function which can be efficiently exploited with OpenMP on 2-way or 4-way SMPs.
2. *Coarse-grained parallelism* at the level of tree alterations and evaluations which can be exploited using MPI and a master–worker scheme.
3. *Job-level parallelism* where multiple phylogenetic analyses on the same dataset with distinct starting trees or multiple bootstrap analyses are performed simultaneously on a cluster.

3.4.1 Job-Level Parallelism

Since implementing job-level parallelism does not represent a very challenging task this issue is omitted. It should be stated however, that this is probably the best way to exploit a parallel computer for real-world biological analyses (including multiple bootstrapping) of large datasets in most practical cases. In order to conduct a biologically "publishable" study, multiple inferences with distinct starting trees and a relatively large number of bootstrap runs should be executed. The typical RAxML

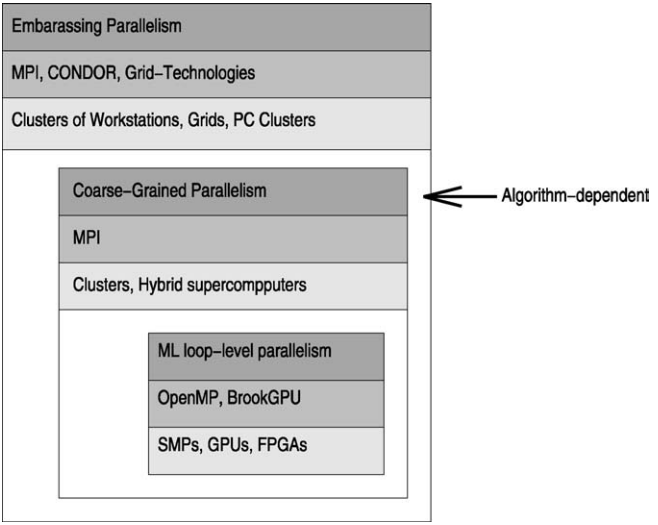

FIG. 18. The three nested sources of parallelism in ML programs.

execution times under elaborate models of nucleotide substitution for trees of 1000–2000 taxa range from 12 to 24 hours on an Opteron CPU. Note that, the dedicated High Performance Computing Version of RAxML-VI (released March 2006) only requires about 40–60 hours in sequential execution mode on 7000–8000 taxon alignments under the reasonably accurate and fast GTR + CAT approximation. In order to provide a useful tool for Biologists this version has also been parallelized with MPI to enable parallel multiple inferences on the original alignment as well as parallel multiple non-parametric bootstraps.

3.4.2 Shared-Memory Parallelism

The exploitation of fine-grained loop-level parallelism is straightforward, since ML programs spend most of their time in the `for`-loops for calculating the likelihood (see Section 3.1). In addition, those loops do not exhibit any dependencies between iteration `i` → `i + 1` such that they can easily be parallelized with OpenMP. As indicated in the pseudo-code below, it suffices to insert a simple OpenMP directive:

```
#pragma omp parallel for private(...)
for(i = 0; i < m; i++)
l_p[i] = f(g(l_q[i], b_pq), g(l_r[i], b_pr));
```

There are several advantages to this approach: The implementation is easy, such that little programming effort (approximately one week) is required to parallelize an ML program with OpenMP. The memory space of the likelihood vectors is equally distributed among processors, such that higher cache efficiency is achieved than in the sequential case, due to the smaller memory footprint. This has partially lead to *significantly* superlinear speedups with the OpenMP version of RAxML [93] on large/long alignments. Figure 19 indicates the speedup values of the OpenMP version of RAxML on a simulated alignment of 300 organisms with a length of $m = 5000$ base pairs for the Xeon, Itanium, and Opteron architectures.

Moreover, modern supercomputer architectures can be exploited in a more efficient manner by a hybrid MPI/OpenMP approach. Finally, it is a very general concept that can easily be applied to other ML phylogeny programs. An unpublished OpenMP parallelization of PHYML by M. Ott and A. Stamatakis yielded comparable—though not superlinear—results. GARLI (Derrick Zwickl, personal communication) and IQPNNI [98] are also currently being parallelized with OpenMP. However, the scalability of this approach is usually limited to 2-way or 4-way SMPs and relatively long alignments due to the granularity of this source of parallelism. However, this type of parallelism represents a good solution for analyses of long multi-gene alignments which are becoming more popular recently. Figure 20 indicates the parallel performance improvement on 1 versus 8 CPUs on one node of

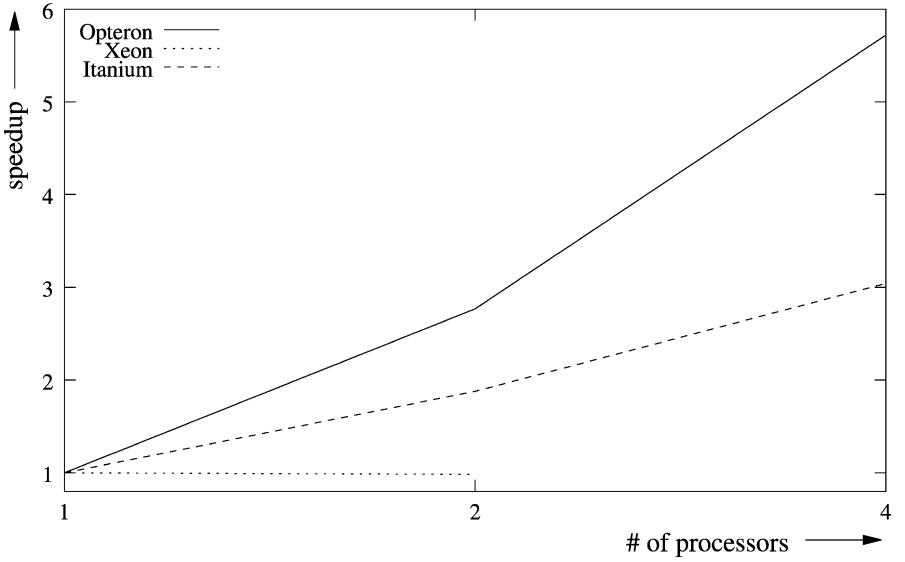

FIG. 19. Speedup of the OpenMP version of RAxML on Xeon, Itanium, and Opteron architectures for a relatively long alignment of 5000 nucleotides.

the CIPRES project (www.phylo.org) cluster located at the San Diego Supercomputing Center for the previously mentioned multi-gene alignment of mammals during the first three iterations of the search algorithm (speedup: 6.74).

Apart from SMPs another interesting hardware platform to exploit loop-level parallelism are GPUs (Graphics Processing Units). Recently, General Purpose computations on GPUs (GPGPU) are becoming more popular due to the availability of improved programming interfaces such a the BrookGPU [99] compiler and runtime implementation. Since GPUs are essentially vector processors the intrinsic fine-grained parallelism of ML programs can be exploited in a very similar way as on SMPs. RAxML has recently been parallelized on a GPU [97] and achieves a highly improved price/performance and power-consumption/performance ratio than on CPUs. Note that, in [97] only one of the main `for`-loops of the program which accounts for approximately 50% of overall execution time has been ported to the GPU. Despite the incomplete porting and the fact that a mid-class GPU (NVIDIA FX 5700LE, price: $75, power consumption: 24 W) and high-end CPU (Pentium 4 3.2 GHz, price: $200, power consumption: $\geqslant$130 W) have been used, a an overall speedup of 1.2 on the GPU has been measured. However, there still exists a relatively large number of technical problems, such as unavailability of double precision arithmetics (RAxML had to be ported to `float`) and insufficient memory capacity

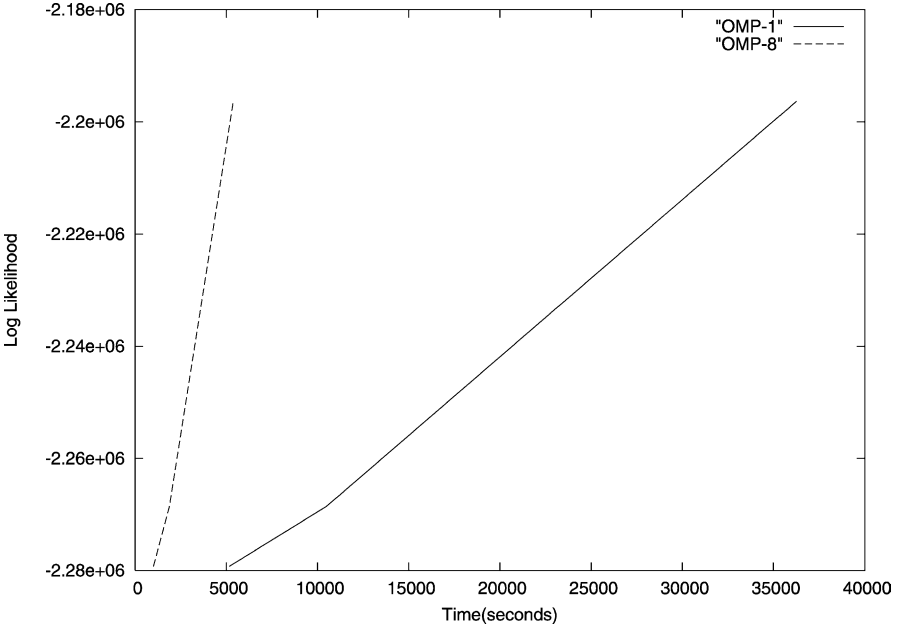

FIG. 20. Run time improvement for the first three iterations of the search algorithm of the OpenMP version of RAxML-VI-HPC on 1 and 8 CPUs on a 51,089 bp long multi-gene alignment of 2182 mammals.

for very large trees (usually up to 512 MB). A natural extension of this work consists in the usage of clusters of GPUs.

3.4.3 *Coarse-Grained Parallelism*

The coarse-grained parallelization of ML phylogeny programs is less straightforward: The parallel efficiency which can be attained depends strongly on the structure of the individual search algorithms. In addition the rate at which improved topologies are encountered has a major impact on parallel efficiency since the tree structure must be continuously updated at all processes. This can result in significant communication overheads.

For example RAxML frequently detects improved topologies during the initial optimization phase of the tree. One iteration of the search algorithm consists in applying a sequence of $2n$ distinct LSR moves (Lazy Subtree Rearrangements, see [57] for details) to the currently best topology t_{best}. If the likelihood of t_{best} is improved by the ith LSR, $i = 1, \ldots, 2n$, the changed topology is kept, i.e. $t_{\text{best}} := t_i$. Thus, one

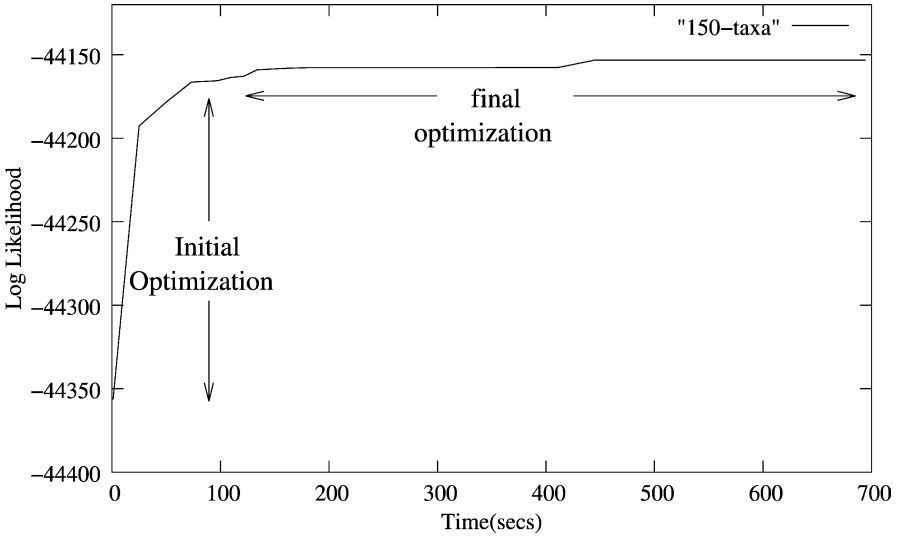

FIG. 21. Typical asymptotic likelihood improvement over time for RAxML on a 150-taxon alignment.

iteration of the sequential algorithm (one iteration of LSRs) generates a sequence of $k \leqslant n$ distinct topologies with improved likelihood values $t_{i_1} \to t_{i_2} \to \cdots \to t_{i_k}$. The likelihood optimization process typically exhibits an asymptotic convergence behavior over time with a steep increase of the likelihood values during the initial optimization phase and an shallow improvement during the final optimization phase (see Fig. 21).

Due to the small execution time of a single LSR even on huge trees, the algorithm can only be parallelized by independently assigning one or more LSR jobs at a time to each worker processes in a master-worker scheme. The main problem consists in breaking up the sequential dependency $t_{i_1} \to t_{i_2} \to \cdots \to t_{i_k}$ of improvements. Since this is very difficult a reasonable approach is to introduce a certain amount of non-determinism. This means that workers will receive topology updates for their local copy of t_{best} detected by other workers with some delay and in a different order. If during the initial optimization phase of RAxML k is relatively large with respect to n, e.g. $k \approx n$ this has a negative impact on the parallel efficiency since a large number of update messages has to be communicated and is delayed. For example, on an alignment with 7769 organisms every second LSR move yielded a topology with an improved likelihood value during the first iteration of the search algorithm. Thus, the mechanism of exchanging topological alterations between workers represents a potential performance bottleneck. The standard string representation of trees (with

parentheses, taxon-names and branch lengths) as used in parallel fastDNAml [83] and an older parallel version of RAxML [92] is becoming inefficient. This is due to the feasibility to compute significantly larger trees caused by recent algorithmic advances. In addition, the starting trees for these large analyses which are usually computed using Neighbor Joining or "quick & dirty" Maximum Parsimony heuristics are worse (in terms of relative difference between the likelihood score of the starting tree and the final tree topology) than on smaller trees. As a consequence improved topologies are detected more frequently during the initial optimization phase with a negative impact on speedup values. Thus, topological changes should be communicated by specifying the actual change only, e.g. remove subtree number i from branch x and insert it into branch y. This can still lead to inconsistencies among the local copies of t_{best} at individual workers but appears to be the only feasible solution for parallelizing the initial optimization phase.

Nonetheless, the final optimization phase which is significantly longer with respect to the total run time of the program is less problematic to parallelize since improved topologies are encountered less frequently. It is important to note, that the above problems mainly concern the parallelization of RAxML but will generally become more prominent as tree sizes grow.

A recent parallelization of IQPNNI [89] with near-optimal relative speedup values demonstrates that these problems are algorithm-dependent. However, as most other programs IQPNNI is currently constrained to tree sizes of approximately 2000 taxa due to memory shortage. The comments about novel solutions which have to be deployed for communicating and updating topologies still hold.

An issue which will surely become important for future HPC ML program development is the distribution of memory among processes: Currently, most implementations hold the entire tree data structure in memory locally at each worker. Given the constant increase of computable tree sizes, and the relatively low main memory per node (1 GB) of current MPP architectures, such as the IBM BlueGene, it will become difficult to hold the complete tree in memory for trees comprising more than 20,000–100,000 taxa.

3.5 Conclusion

Due to the significant progress, which has been achieved by the introduction of novel search algorithms for ML-based phylogenetic inference, analyses of huge phylogenies comprising several hundreds or even thousands of taxa have now become feasible. However, the performance of ML phylogeny programs is increasingly limited by rarely documented and published technical implementation issues. Thus, an at least partial paradigm shift towards technical issues is required in order to advance

the field and to enable inference of larger trees with the ultimate, though still distant, goal to compute the tree-of-life.

As an example for the necessity of a paradigm shift one can consider the recent improvements to RAxML: The significant (unpublished) speedups for sequential RAxML-VI over sequential RAxML-V, of 1.66 on 1000 taxa over 30 on 4000 taxa up to 67 on 25,000 taxa, have been attained by very simple technical optimizations of the code.[1] The potential for these optimizations has only been realized by the author who has been working on the RAxML-code for almost 4 years after the respective paradigm shift.

To this end, the current Section covered some of those rarely documented but increasingly important technical issues and summarizes how MPP, SMP, hybrid supercomputer, and GPU architectures can be used to infer huge trees.

ACKNOWLEDGEMENTS

Bader's research discussed in this chapter has been supported in part by NSF Grants CAREER ACI-00-93039, CCF-0611589, DBI-0420513, ITR ACI-00-81404, ITR EIA-01-21377, Biocomplexity DEB-01-20709, and ITR EF/BIO 03-31654; and DARPA Contract NBCH30390004.

REFERENCES

[1] Bader D., Moret B., Vawter L., "Industrial applications of high-performance computing for phylogeny reconstruction", in: Siegel H. (Ed.), *Proc. SPIE Commercial Applications for High-Performance Computing, Denver, CO*, vol. 4528, SPIE, Bellingham, WA, 2001, pp. 159–168.

[2] Moret B., Wyman S., Bader D., Warnow T., Yan M., "A new implementation and detailed study of breakpoint analysis", in: *Proc. 6th Pacific Symp. Biocomputing*, *PSB 2001, Hawaii*, 2001, pp. 583–594.

[3] Yan M., "High performance algorithms for phylogeny reconstruction with maximum parsimony", PhD thesis, Electrical and Computer Engineering Department, University of New Mexico, Albuquerque, NM, 2004.

[4] Yan M., Bader D.A., "Fast character optimization in parsimony phylogeny reconstruction", Technical report, Electrical and Computer Engineering Department, The University of New Mexico, Albuquerque, NM, 2003.

[5] Caprara A., "Formulations and hardness of multiple sorting by reversals", in: *3rd Ann. Internat. Conf. Computational Molecular Biology*, *RECOMB99, Lyon, France*, ACM, New York, 1999.

[1] Performance results and datasets used for RAxML-VI are available on-line at www.ics.foth.gr/~stamatak (material frame).

[6] Pe'er I., Shamir R., "The median problems for breakpoints are NP-complete", Technical Report 71, Electronic Colloquium on Computational Complexity, 1998.
[7] Swofford D., Olsen G., Waddell P., Hillis D., "Phylogenetic inference", in: Hillis A., Moritz C., Mable B. (Eds.), *Molecular Systematics*, Sinauer Associates, Sunderland, MA, 1996, pp. 407–514.
[8] Nei M., Kumar S., *Molecular Evolution and Phylogenetics*, Oxford Univ. Press, Oxford, UK, 2000.
[9] Faith D., "Distance method and the approximation of most-parsimonious trees", *Systematic Zoology* **34** (1985) 312–325.
[10] Farris J., "Estimating phylogenetic trees from distance matrices", *Amer. Naturalist* **106** (1972) 645–668.
[11] Li W.H., "Simple method for constructing phylogenetic trees from distance matrices", *Proc. Natl. Acad. Sci. USA* **78** (1981) 1085–1089.
[12] Saitou N., Nei M., "The neighbor-joining method: A new method for reconstruction of phylogenetic trees", *Mol. Biol. Evol.* **4** (1987) 406–425.
[13] Studier J., Keppler K., "A note on the neighbor-joining method of Saitou and Nei", *Mol. Biol. Evol.* **5** (1988) 729–731.
[14] Rice K., Warnow T., "Parsimony is hard to beat", in: *Computing and Combinatorics*, 1997, pp. 124–133.
[15] Hendy M., Penny D., "Branch and bound algorithms to determine minimal evolutionary trees", *Math. Biosci.* **59** (1982) 277–290.
[16] Fitch W., "Toward defining the course of evolution: Minimal change for a specific tree topology", *Systematic Zoology* **20** (1971) 406–416.
[17] Purdom Jr., Bradford P., Tamura K., Kumar S., "Single column discrepancy and dynamic max-mini optimization for quickly finding the most parsimonious evolutionary trees", *Bioinformatics* **2** (16) (2000) 140–151.
[18] Eck R., Dayhoff M., *Atlas of Protein Sequence and Structure*, National Biomedical Research Foundation, Silver Spring, MD, 1966.
[19] Benaïchouche M., Cung V., Dowaji S., Cun B., Mautor T., Roucairol C., "Building a parallel branch and bound library", in: Ferreira A., Pardalos P. (Eds.), *Solving Combinatorial Optimization Problem in Parallel: Methods and Techniques*, Springer-Verlag, Berlin, 1996, pp. 201–231.
[20] Swofford D., Begle D., *PAUP: Phylogenetic Analysis Using Parsimony*, Sinauer Associates, Sunderland, MA, 1993.
[21] Felsenstein J., "PHYLIP—phylogeny inference package (version 3.2)", *Cladistics* **5** (1989) 164–166.
[22] Goloboff P., "Analyzing large data sets in reasonable times: Solutions for composite optima", *Cladistics* **15** (1999) 415–428.
[23] Nixon K., "The parsimony ratchet, a new method for rapid parsimony analysis", *Cladistics* **15** (1999) 407–414.
[24] Sankoff D., Blanchette M., "Multiple genome rearrangement and breakpoint phylogeny", *J. Comput. Biol.* **5** (1998) 555–570.

[25] Cosner M., Jansen R., Moret B., Raubeson L., Wang L.S., Warnow T., Wyman S., "An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae", in: Sankoff D., Nadeau J. (Eds.), *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Map Alignment, and the Evolution of Gene Families*, Kluwer Academic, Dordrecht, Netherlands, 2000, pp. 99–121.

[26] Bader D., Moret B., "GRAPPA runs in record time", *HPCwire* **9** (47) (2000).

[27] Giribet G., "A review of tnt: Tree analysis using new technology: Version 1.0, beta test v. 0.2". Program and documentation available at http://www.zmuc.dk/public/phylogeny/tnt/;
Goloboff P.A., Farris J.S., Nixon K., "Instituto Miguel Lillo, San Miguel de Tucuman, Argentina", *Syst. Biol.* (2005) 176–178.

[28] Meier R., Ali F., "The newest kid on the parsimony block: Tnt (tree analysis using new technology", *Syst. Entomol.* **30** (2005) 179–182.

[29] Goloboff P., "Analyzing large data sets in reasonable times: solution for composite optima", *Cladistics* **15** (1999) 415–428.

[30] Swofford D.L., *PAUP*: Phylogenetic Analysis Using Parsimony (and Other Methods), Version 4.0*, Sinauer Associates, Underland, MA, 1996.

[31] Stamatakis A., Ludwig T., Meier H., "Raxml-iii: A fast program for maximum likelihood-based inference of large phylogenetic trees", *Bioinformatics* **21** (4) (2005) 456–463.

[32] Huson D., Nettles S., Warnow T., "Disk-covering, a fast-converging method for phylogenetic tree reconstruction", *J. Comput. Biol.* **6** (1999) 369–386.

[33] Huson D., Vawter L., Warnow T., "Solving large scale phylogenetic problems using DCM2", in: *Proc. 7th Internat. Conf. on Intelligent Systems for Molecular Biology, ISMB'99*, AAAI Press, Menlo Park, CA, 1999, pp. 118–129.

[34] Roshan U., Moret B.M.E., Warnow T., Williams T.L., "Rec-i-dcm3: a fast algorithmic technique for reconstructing large phylogenetic trees", in: *Proc. of CSB04, Stanford, CA*, 2004.

[35] Roshan U., "Algorithmic techniques for improving the speed and accuracy of phylogenetic methods", PhD thesis, The University of Texas at Austin, 2004.

[36] Roshan U., Moret B.M.E., Williams T.L., Warnow T., "Performance of supertree methods on various dataset decompositions", in: Bininda-Emonds O.R.P. (Ed.), *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, in: Dress A. (Ed.), *Computational Biology*, vol. 3, Kluwer Academic, Dordrecht/Norwell, MA, 2004, pp. 301–328.

[37] Nakhleh L., Roshan U., St. John K., Sun J., Warnow T., "Designing fast converging phylogenetic methods", in: *Proc. 9th Internat. Conf. on Intelligent Systems for Molecular Biology, ISMB'01*, in: *Bioinformatics*, vol. 17, Oxford Univ. Press, Oxford, UK, 2001, pp. S190–S198.

[38] Nakhleh L., Moret B., Roshan U., John K.S., Warnow T., "The accuracy of fast phylogenetic methods for large datasets", in: *Proc. 7th Pacific Symp. Biocomputing, PSB'2002*, World Scientific Publ., Singapore, 2002, pp. 211–222.

[39] Nakhleh L., Roshan U., St. John K., Sun J., Warnow T., "The performance of phylogenetic methods on trees of bounded diameter", in: *Proc. of WABI'01*, in: *Lecture Notes in Comput. Sci.*, vol. 2149, Springer-Verlag, Berlin, 2001, pp. 214–226.

[40] Moret B., Roshan U., Warnow T., "Sequence length requirements for phylogenetic methods", in: *Proc. of WABI'02*, 2002, pp. 343–356.
[41] Golumbic M., *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, San Diego, CA, 1980.
[42] Hoos H.H., Stutzle T., *Stochastic Local Search: Foundations and Applications*, Morgan Kaufmann, San Francisco, CA, 2004.
[43] Du Z., Stamatakis A., Lin F., Roshan U., Nakhleh L., "Parallel divide-and-conquer phylogeny reconstruction by maximum likelihood". Accepted to the *2005 International Conference on High Performance Computing and Communications*, pending publication in proceedings.
[44] Stewart C., Hart D., Berry D., Olsen G., Wernert E., Fischer W., "Parallel implementation and performance of fastdnaml—a program for maximum likelihood phylogenetic inference", in: *Proceedings of the 14th IEEE/ACM Supercomputing Conference, SC2001*, 2001.
[45] Ludwig W., Strunk O., Westram R., Richter L., Meier H., Yadhukumar, Buchner A., Lai T., Steppi S., Jobb G., Fvrster W., Brettske I., Gerber S., Ginhart A.W., Gross O., Grumann S., Hermann S., Jost R., Kvnig A., Liss T., Lubmann R., May M., Nonhoff B., Reichel B., Strehlow R., Stamatakis A., Stuckman N., Vilbig A., Lenke M., Ludwig T., Bode A., Schleifer K.H., "Arb: a software environment for sequence data", *Nucl. Acids Res.* **32** (4) (2004) 1363–1371.
[46] Vinh L., Haeseler A., "Iqpnni: Moving fast through tree space and stopping in time", *Mol. Biol. Evol.* **21** (2004) 1565–1571.
[47] Maidak B., Cole J., Lilburn T. , Parker C.T. Jr, Saxman P., Farris R., Garrity G., Olsen G., Schmidt T., Tiedje J., "The RDP-II (Ribosomal Database Project)", *Nucl. Acids Res.* **29** (1) (2001) 173–174.
[48] Lipscomb D., Farris J., Kallersjo M., Tehler A., "Support, ribosomal sequences and the phylogeny of the eukaryotes", *Cladistics* **14** (1998) 303–338.
[49] Rice K., Donoghue M., Olmstead R., "Analyzing large datasets: *rbcL* 500 revisited", *Syst. Biol.* **46** (3) (1997) 554–563.
[50] Soltis D.E., Soltis P.S., Chase M.W., Mort M.E., Albach D.C., Zanis M., Savolainen V., Hahn W.H., Hoot S.B., Fay M.F., Axtell M., Swensen S.M., Prince L.M., Kress W.J., Nixon K.C., Farris J.S., "Angiosperm phylogeny inferred from 18s rDNA, *rbcL*, and *atpB* sequences", *Botanical J. Linnean Soc.* **133** (2000) 381–461.
[51] Johnson K.P., "Taxon sampling and the phylogenetic position of passeriformes: Evidence from 916 avian cytochrome b sequences", *Syst. Biol.* **50** (1) (2001) 128–136.
[52] Felsenstein J., "Phylip (phylogeny inference package) version 3.6". Distributed by the author, Department of Genome Sciences, University of Washington, Seattle, 2004.
[53] Swofford D.L., Olsen G.J., "Phylogeny reconstruction", in: Hillis D., Moritz C., Marble B.K. (Eds.), *Molecular Systematics*, second ed., Sinauer Associates, Sunderland, MA, 1996, pp. 407–514.
[54] Felsenstein J., "Evolutionary trees from DNA sequences: A maximum likelihood approach", *J. Mol. Evol.* **17** (1981) 368–376.
[55] Ley R., Backhed F., Turnbaugh P., Lozupone C., Knight R., Gordon J., "Obesity alters gut microbial ecology", *Proc. Natl. Acad. Sci. USA* **102** (31) (2005) 11070–11075.

[56] Chor B., Tuller T., "Maximum likelihood of evolutionary trees is hard", in: *Proc. of RECOMB05*, 2005.

[57] Stamatakis A., Ludwig T., Meier H., "Raxml-iii: A fast program for maximum likelihood-based inference of large phylogenetic trees", *Bioinformatics* **21** (4) (2005) 456–463.

[58] Guindon S., Gascuel O., "A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood", *Syst. Biol.* **52** (5) (2003) 696–704.

[59] Brauer M., Holder M., Dries L., Zwickl D., Lewis P., Hillis D., "Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference", *Mol. Biol. Evol.* **19** (2002) 1717–1726.

[60] Lemmon A., Milinkovitch M., "The metapopulation genetic algorithm: An efficient solution for the problem of large phylogeny estimation", *Proc. Natl. Acad. Sci.* **99** (2001) 10516–10521.

[61] Jobb G., Haeseler A., Strimmer K., "Treefinder: A powerful graphical analysis environment for molecular phylogenetics", *BMC Evol. Biol.* **4** (2004).

[62] Kosakovsky-Pond S., Muse S., "Column sorting: Rapid calculation of the phylogenetic likelihood function", *Syst. Biol.* **53** (5) (2004) 685–692.

[63] Stamatakis A., Ludwig T., Meier H., Wolf M. "Accelerating parallel maximum likelihood-based phylogenetic tree calculations using subtree equality vectors", in: *Proc. of 15th IEEE/ACM Supercomputing Conference, SC2002*, 2002.

[64] Swofford D., Olsen G., "Phylogeny reconstruction", in: Hillis D., Moritz C. (Eds.), *Molecular Systematics*, Sinauer Associates, Sunderland, MA, 1990, pp. 411–501.

[65] Lanave C., Preparata G., Saccone C., Serio G., "A new method for calculating evolutionary substitution rates", *J. Mol. Evol.* **20** (1984) 86–93.

[66] Rodriguez F., Oliver J., Marin A., Medina J., "The general stochastic model of nucleotide substitution", *J. Theor. Biol.* **142** (1990) 485–501.

[67] Jukes T., Cantor C. III, *Evolution of Protein Molecules*, Academic Press, New York, 1969, pp. 21–132.

[68] Hasegawa M., Kishino H., Yano T., "Dating of the human-ape splitting by a molecular clock of mitochondrial DNA", *J. Mol. Evol.* **22** (1985) 160–174.

[69] Posada D., Crandall K., "Modeltest: testing the model of DNA substitution", *Bioinformatics* **14** (9) (1998) 817–818.

[70] Olsen G., Matsuda H., Hagstrom R., Overbeek R., "fastdnaml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood", *Comput. Appl. Biosci.* **20** (1994) 41–48.

[71] Yang Z., "Among-site rate variation and its impact on phylogenetic analyses", *Trends Ecol. Evol.* **11** (1996) 367–372.

[72] Olsen G., Pracht S., Overbeek R., "Dnarates distribution", http://geta.life.uiuc.edu/~gary/programs/DNArates.html, unpublished, 1998.

[73] Meyer S., v. Haeseler A., "Identifying site-specific substitution rates", *Mol. Biol. Evol.* **20** (2003) 182–189.

[74] Yang Z., "Maximum likelihood phylogenetic estimation from DNA sequences with variable rates over sites", *J. Mol. Evol.* **39** (1994) 306–314.

[75] Stamatakis A., "Phylogenetic models of rate heterogeneity: A high performance computing perspective", in: *Proc. of IPDPS2006, Rhodos, Greece*, 2006.
[76] Zwickl D., "Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion", PhD thesis, University of Texas at Austin, 2006.
[77] Hordijk W., Gascuel O., "Improving the efficiency of spr moves in phylogenetic tree search methods based on maximum likelihood", *Bioinformatics* (2005).
[78] Swofford D., *PAUP*: Phylogenetic Analysis Using Parsimony (and Other Methods), Version 4.0*, Sinauer Associates, Underland, MA, 1996.
[79] Strimmer K., Haeseler A., "Quartet puzzling: A maximum-likelihood method for reconstructing tree topologies", *Mol. Biol. Evol.* **13** (1996) 964–969.
[80] Stamatakis A. "An efficient program for phylogenetic inference using simulated annealing", in: *Proc. of IPDPS2005, Denver, CO*, 2005.
[81] Salter L., Pearl D., "A stochastic search strategy for estimation of maximum likelihood phylogenetic trees", *Syst. Biol.* **50** (1) (2001) 7–17.
[82] Barker D., "Lvb: Parsimony and simulated annealing in the search for phylogenetic trees", *Bioinformatics* **20** (2004) 274–275.
[83] Stewart C., Hart D., Berry D., Olsen G., Wernert E., Fischer W., "Parallel implementation and performance of fastdnaml—a program for maximum likelihood phylogenetic inference", in: *Proc. of SC2001*, 2001.
[84] Hart D., Grover D., Liggett M., Repasky R., Shields C., Simms S., Sweeny A., Wang P., "Distributed parallel computing using windows desktop system", in: *Proc. of CLADE*, 2003.
[85] Keane T., Naughton T., Travers S., McInerney J., McCormack G., "Dprml: Distributed phylogeny reconstruction by maximum likelihood", *Bioinformatics* **21** (7) (2005) 969–974.
[86] Wolf M., Easteal S., Kahn M., McKay B., Jermiin L., "Trexml: A maximum likelihood program for extensive tree-space exploration", *Bioinformatics* **16** (4) (2000) 383–394.
[87] Zhou B., Till M., Zomaya A., Jermiin L., "Parallel implementation of maximum likelihood methods for phylogenetic analysis", in: *Proc. of IPDPS2004*, 2004.
[88] Schmidt H., Strimmer K., Vingron M., Haeseler A., "Tree-puzzle: maximum likelihood phylogenetic analysis using quartets and parallel computing", *Bioinformatics* **18** (2002) 502–504.
[89] Minh B., Vinh L., Haeseler A., Schmidt H., "piqpnni—parallel reconstruction of large maximum likelihood phylogenies", *Bioinformatics* (2005).
[90] Du Z., Stamatakis A., Lin F., Roshan U., Nakhleh L., "Parallel divide-and-conquer phylogeny reconstruction by maximum likelihood", in: *Proc. of HPCC-05*, 2005, pp. 776–785.
[91] Stamatakis A., Lindermeier M., Ott M., Ludwig T., Meier H., "Draxml@home: A distributed program for computation of large phylogenetic trees", *Future Generation Comput. Syst.* **51** (5) (2005) 725–730.
[92] Stamatakis A., Ludwig T., Meier H., "Parallel inference of a 10.000-taxon phylogeny with maximum likelihood", in: *Proc. of Euro-Par2004*, 2004, pp. 997–1004.

[93] Stamatakis A., Ott M., Ludwig T., "Raxml-omp: An efficient program for phylogenetic inference on SMPs", in: *Proc. of PaCT05*, 2005, pp. 288–302.
[94] Huelsenbeck J., Ronquist F., "Mrbayes: Bayesian inference of phylogenetic trees", *Bioinformatics* **17** (2001) 754–755.
[95] Press W., Teukolsky S., Vetterling W., Flannery B., *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge Univ. Press, New York, 1992.
[96] Yang Z., "Maximum likelihood estimation on large phylogenies and analysis of adaptive evolution in human influenza virus a", *J. Mol. Evol.* **51** (2000) 423–432.
[97] Charalambous M., Trancoso P., Stamatakis A., "Initial experiences porting a bioinformatics application to a graphics processor", in: *Proceedings of the 10th Panhellenic Conference on Informatics, PCI 2005*, 2005, pp. 415–425.
[98] Minh B., Schmidt H., Haeseler A., "Large maximum likelihood trees", Technical report, John von Neumann Institute for Computing, Jülich, Germany, 2006.
[99] Buck I., Foley T., Horn D., Sugerman J., Hanrahan P., Houston M., Fatahalian K., "Brookgpu website", http://graphics.stanford.edu/projects/brookgpu/index.html, 2005.