# Contents

**1 High-Performance Phylogeny Reconstruction Under Maximum Parsimony 1**

*Tiffani L. Williams, David A. Bader, Bernard M.E. Moret, and Mi Yan*

1.1 Introduction 1

1.2 Maximum Parsimony 10

1.3 Exact MP: Parallel Branch & Bound 17

1.4 MP Heuristics: Disk-Covering Methods 25

1.5 Summary and Open Problems 36

References 38

# 1 High-Performance Phylogeny Reconstruction Under Maximum Parsimony

TIFFANI L. WILLIAMS[†], DAVID A. BADER[‡], BERNARD M.E. MORET[§], and MI YAN[†]

## 1.1 INTRODUCTION

The similarity of the molecular matter of the organisms on Earth suggest that they all share a common ancestor. Thus any set of species is related, and this relationship is called a phylogeny. The links (or evolutionary relationships) among a set of organisms (or taxa) form a phylogenetic tree, where modern organisms are placed at the leaves and ancestral organisms occupy internal nodes, with the edges of the tree denoting evolutionary relationships. Scientists are interested in evolutionary trees for the usual reasons of scientific curiosity. However, phylogenetic analysis is not just an academic exercise. Phylogenies are the organizing principle for most biological knowledge. As such, they are a crucial tool in identifying emerg-

[†] Department of Computer Science, Texas A&M University
[‡] Department of Electrical and Computer Engineering, University of New Mexico
[§] Department of Computer Science, University of New Mexico

ing diseases, predicting disease outbreaks, and protecting ecosystems from invasive species [Bader et al., 2001, Cracraft, 2002]. The greatest impact of phylogenetics will be reconstructing the Tree of Life, the evolutionary history of all-known organisms. The precise number of organisms that exist is not known; estimates range from 10-100 million species. Today only about 1.7 million species are known.

Given the enormous implications of phylogenetic analysis, reconstructing the evolutionary history of a set of taxa is a very difficult problem. For $n$ organisms, there are $(2n-5)(2n-3)\cdots(5)(3)$ distinct binary trees—each a possible hypothesis for the "true" evolutionary history. For example, there are over 13 billion possible trees for 13 taxa. Since the size of the tree space increases exponentially with the number of taxa, it is impossible to consider all of the possible trees within a reasonable time frame. Most phylogenetic methods limit themselves to exhaustive searches on small datasets and heuristic strategies for larger datasets. Another difficulty lies in accessing the accuracy of the reconstructed tree. Short of traveling back into time, there is no way of determining whether the proposed evolutionary history is 100% correct.

### 1.1.1 Phylogenetic data

Early evolutionary trees were built by examining the similarities and differences of form and structure of the the organisms of interest. Such an approach relies on identifying morphological characters (i.e., presence of wings) and classifying organisms based on the presence or absence of these features. Species are represented by binary sequences corresponding to the morphological data. Each bit corresponds to a char-

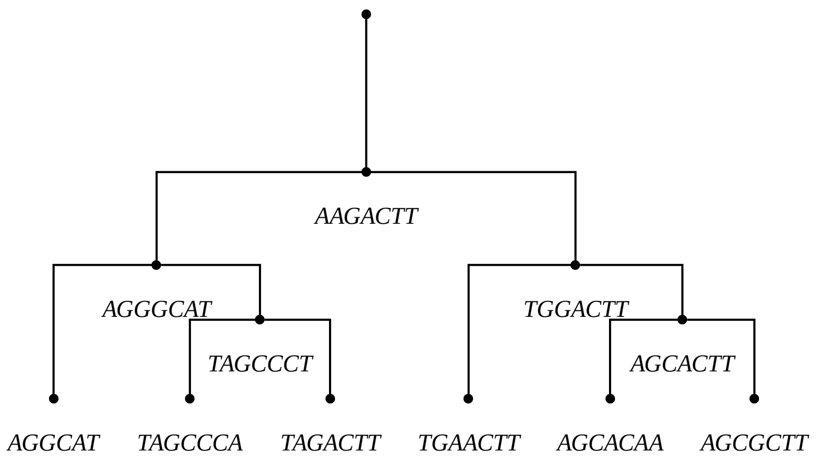


**Fig. 1.1** Evolving sequences down a fixed tree.

acter. If a species has a given feature, the corresponding bit is set to one; otherwise, it is zero. Yet, relying solely on morphological characters can be a major source of phylogenetic error.

With the advent of molecular data, scientists hope to avoid the problems with morphological criteria of relatedness. Today, most trees are built exclusively from molecular sequences. In sequence data, characters are individual positions (or sites) in the string, where characters can assume one of four states for nucleotides (A,C,G,T) or one of 20 states for amino-acids. Sequence evolution is studied under a simplifying assumption that each site evolves independently. Data evolves through point mutations (i.e., changes in the state of a character), plus insertions (including duplications) and deletions.

Figure 1.1 shows a simple evolutionary history, from the ancestral sequence at the root (AAGACTT) to modern sequences at the leaves, with evolutionary events occurring on each edge. Note that this history is incomplete, as it does not detail the events that have taken place along each edge of the tree. Thus, while one might reasonably conclude that, in order to reach the leftmost leaf, labeled AGGCAT, from its parent, labeled AGGGCAT, one should infer the deletion of one nucleotide (one of the three G's in the parent), a more complex scenario may in fact have unfolded. If one were to compare the leftmost leaf with the rightmost one, labeled AGCGCTT, one could account for the difference with two changes: starting with AGGCAT, insert a C between the two G's to obtain AGCGCAT, then mutate the penultimate A into a T. Yet the tree itself indicates that the change occurred in a far more complex manner: the path between these two leaves in the tree goes through the series of sequences

$$\text{AGGCAT} \leftrightarrow \text{AGGGCAT} \leftrightarrow \text{AAGACTT} \leftrightarrow \text{TGGACTT} \leftrightarrow \text{AGCACTT} \rightarrow \text{AGCGCTT}$$

and each arrow in this series indicates at least one evolutionary event.

Obtaining sequence data is relatively easy given that large amounts of sequence data are easily attainable from databases such as GenBank, along with search tools (such as BLAST). However, "raw" sequence data must first be refined into a format suitable for use in a phylogenetic analysis. The refinement process is composed of the following four steps.

1. Identifying *homologous* genes (i.e., genes that have evolved from a common ancestral gene—and most likely fulfill the same function in each organism) across the organisms of interest.

2. Retrieving followed by aligning the sequences of these genes across the entire set of organism, in order to identify gaps (corresponding to insertions or deletions) and matches or mutations.

3. Deciding whether to use all available data at once for a *combined* analysis or use each gene separately and *reconcile* the resulting trees.

4. Applying a phylogenetic method to the aligned sequence data (see Section 1.1.2).

Many packages requiring sequence data are available to reconstruct phylogenetic trees such as PAUP* [Swofford, 2002], Phylip [Felsenstein, 89], MrBayes [Huelsenbeck and Ronquist, 2001], and TNT [Goloboff, 1999]. These packages are available either freely or for a modest fee, are in widespread use, and have provided biologists with satisfactory results on many datasets.

Phylogenetic inference based on gene order data provides an alternative to using sequence data [Moret et al., 2005, Moret and Warnow, 2005]. Gene order data is based on the structural arrangement of genes in an organism's entire genome. Hence, the gene tree/species tree problem (i.e., the evolution of any given of any given gene need not be identical to that of the organism) is avoided when using gene order data. Gene order data is sparsely available in comparison to sequence data. Consequently, most tree reconstruction efforts have focused on the reconstruction of phylogenetic trees based on sequence data.

### 1.1.2 Phylogenetic methods

There are a plethora of methods—each accounting for some aspect of the evolutionary process— that can be used to infer a phylogenetic tree. The methods can be divided into two broad categories: *distance methods* transform the sequence data into a numerical representation of the data whereas *criteria-based methods* rely on optimality criteria to score the tree based on the individual contribution of the characters in the sequence data.

***1.1.2.1 Distance methods*** Nucleotide sequence similarities can be converted to sequence distances in order to infer a phylogenetic tree. For $n$ sequences, pairwise distances are calculated to produce a $n \times n$ distance matrix. One could compute the Hamming distance between two sequences as an estimate of their evolutionary distance. However, the Hamming distance is an underestimate of the true genetic distance between a pair of sequences. If mutations occur at random, there will be a certain number of positions with *silent mutations*–changes that are subsequently reversed in the course of evolution, leaving no trace in modern organisms (see Figure 1.2). Therefore, in distance-based methods, one estimates the number of substitutions that have actually occurred by applying a specific *evolutionary model*, which makes assumptions about the nature of evolutionary changes. For example, the Jukes-Cantor model simply assumes that when a base changes, it is equally-likely to change to each of the three alternatives [Jukes, 1969]. Other models, such as Kimura's two-parameter model [Kimura, 1980], provide additional parameters to

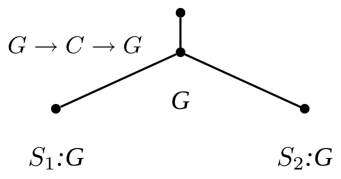


**Fig. 1.2** Silent mutation. The $C$ mutation is unobserved in $S_1$. There are no mutations from the root node to $S_2$. The Hamming distance between sequences $S_1$ and $S_2$ is 0, which is an underestimation of the pattern of evolution ($G \rightarrow C \rightarrow G \rightarrow G$) from $S_1$ to $S_2$.

compute to compute the probability of change from a given state to another given state.

Distance-based methods build the search for the "true" tree into the algorithm, thus returning a unique final topology for a distance matrix associated with a given set of sequences. Neighbor-Joining (NJ)—the most popular distance-based algorithm [Saitou and Nei, 1987]—begins with each organism in its own subtree. The algorithm joins the pair with the minimum distance, making a subtree whose root replaces the two chosen taxa in the matrix. Distances are recalculated based on this new node, and the joining continues until three nodes remain. These nodes are joined to form an unrooted binary tree. Appealing features of NJ are its simplicity and speed—it runs in $O(n^3)$ time. Other distance methods include refinements of NJ such as BioNJ [Gascuel, 1997] and Weighbor [Bruno et al., 2000].

***1.1.2.2 Criteria-based methods*** Criteria-based methods explicitly rank the tree topologies by defining an objective function to score the trees. Tree scores allow any two or more trees to be ranked according to the chosen optimality criterion.

Unlike distance-based approaches, there are many possible solutions for a given set of sequences. Hence, there is an explicit search for the "optimal" tree. Maximum parsimony (MP) and maximum likelihood (ML) are two of the major optimization problems in phylogeny reconstruction, but both are quite difficult to solve (MP is NP-hard [Foulds and Graham, 1982], and ML harder in practice.) We briefly discuss each in turn below. MP is discussed more thoroughly in Section 1.2.

*Maximum parsimony* An intuitive approach for ranking phylogenetic trees is counting the total number of mutations required to explain all of the observed character sequences. MP attempts to minimize this score following the philosophy of Occam's razor—the simplest explanation of the data is preferred. Under MP, a total cost is assigned to each tree, and the optimal tree (i.e., the most parsimonious tree) is defined as the one with the smallest total cost. In this approach, a unit cost is given for each nucleotide substitution. A central step in the procedure is allocating sequences to the internal nodes in the tree. For any set of sequence allocations, the total cost of the tree is the sum of the costs of the various edges, where the cost of joining two internal nodes, or an internal node and a leaf, is the number of substitutions needed to move from the sequence at one to the sequence at the other (i.e., the Hamming distance). Many software packages implement MP heuristics, among them the most popular are PAUP* [Swofford, 2002], Phylip [Felsenstein, 89], and TNT [Goloboff, 1999].

*Likelihood methods* Likelihood methods require the specification of an evolutionary model. For a given phylogenetic arrangement, the question is: what is the likelihood that evolution under the specified parameters will produce the observed nucleotide sequences? For a given evolutionary hypothesis, the likelihood of the observed change is computed for each position, and then the product of the likelihoods is expressed as a distance or branch length between the sequences. The parameters are then varied, and the combination with the highest likelihood is accepted. This procedure is then repeated for another arrangement, the two topologies compared, and the one with the highest likelihood selected. The selective process is continued until an arrangement is found with the combined maximum likelihood of both an evolutionary hypothesis and a topology. Finding the best tree under maximum likelihood is the most computationally demanding of the methods discussed here.

Like MP, ML is an optimization problem. ML seeks the tree and associated model parameter values that maximizes the probability of producing the given set of sequences. ML thus depends explicitly on an assumed model of evolution. For example, the ML problem under the Jukes-Cantor model needs to estimate one parameter (the substitution probability) for each edge of the tree, while under the General Markov model 12 parameters must be estimated on each edge. Unlike MP, scoring a fixed tree, cannot be done in polynomial time for ML [Steel, 1994], whereas it is easily accomplished in linear time for MP [Fitch, 1971]. Various software packages provide heuristics for ML, include PAUP* [Swofford, 2002], Phylip [Felsenstein, 89], fastDNAml [Olsen et al., 1994], and PHYML [Guindon and Gascuel, 2003].

Bayesian methods deserve a special mention among likelihood-based approaches; they compute the posterior probability that the observed data would have been produced by various trees (in contrast to a true maximum likelihood method, which computes the probability that a fixed tree would produce various kinds of data at its leaves). Their implementation with Markov chain Monte-Carlo (MCMC) algorithms often run significantly faster than pure ML methods. Moreover, the moves through state space can be designed to enhance convergence rates and speed up execution. MrBayes [Huelsenbeck and Ronquist, 2001] is the most popular software package for reconstructing trees based on Bayesian analysis.

### 1.1.3 Large-scale phylogenies

The remainder of this paper considers high-performance approaches to inferring trees under maximum parsimony. Our reasons for focusing on MP are two-fold. First, MP remains the major approach by which phylogenies are reconstructed. A survey of 882 phylogenetic analyses published in 76 journals revealed that 60% of the phylogenies were constructed using MP heuristics [Sanderson et al., 1993]. Secondly, ML methods are to slow to infer trees for large datasets($>$ 1000 sequences). Although distance methods can handle large datasets, studies show they produce trees with very high topological error [Moret et al., 2002, Nakhleh et al., 2002, Nakhleh et al., 2001].

## 1.2 MAXIMUM PARSIMONY

Maximum parsimony is an optimization problem for inferring phylogenetic trees, where each of the taxa in the input is represented by a string over some alphabet. The input consists of a set $S$ of $n$ strings over a fixed alphabet $\Sigma$, where

$\Sigma = \{A, C, G, T\}$ represents the set of four nucleotides. $\Sigma$ could also represent the set of twenty amino-acids. The elements of $\Sigma$ are also called states (or characters). We assume that the sequence data has been properly prepared; particularly, the sequences are already aligned so that all sequences are of length $k$. Positions within the sequences are sometimes called sites.

Formally, given two sequences $a$ and $b$ of length $k$, the *Hamming distance* between them is defined as $H(a, b) = |\{i : a_i \neq b_i\}|$. Let $T$ be a tree whose nodes are labeled by sequences of length $k$ over $\Sigma$, and let $H(e)$ denote the Hamming distance of the sequences at each endpoint of $e$. Then the *parsimony length* of the tree $T$ is $\sum_{e \in E(T)} H(e)$. The MP problem seeks the tree $T$ with the minimum length; this is the same as seeking the tree with the smallest number of point mutations for the data. MP is an NP-hard problem [Foulds and Graham, 1982], but the problem of assigning sequences to internal nodes of a fixed leaf-labeled tree is polynomial [Fitch, 1971].

### 1.2.1 Scoring a fixed tree

Fitch's algorithm can be applied to calculate the parsimony score of a fixed tree [Fitch, 1977] . First, we define the possible states for each of the internal nodes that minimize the score. Let $S_v \subseteq \Sigma$ denote the set of state assignments for node $v$. We assume that $T$ is binary and the children of $v$ are $x$ and $y$. If $v$ is a leaf, then $S_v$ is simply the state of $v$. If $v$ is an internal node, then it's state is based on the state of its two children $x$ and $y$ (i.e., $S_x$ and $S_y$). If $S_x \cap S_y \neq \oslash$, then $S_v = S_x \cap S_y$. Otherwise, $S_v = S_x \cup S_y$.

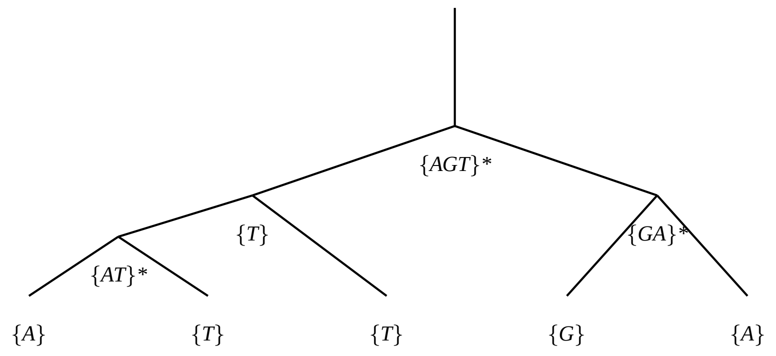


**Fig. 1.3** Fitch's algorithm applied to single site. Sets denoted by an * increase the parsimony score by one. Thus, the parsimony score of this tree is 3.

Using a postorder traversal, the above equalities allow us to compute $S_v$ for every node $v$ in $T$, from the bottom up (see Figure 1.3). Moreover, the optimal cost (or maximum parsimony score) of $T$ can be calculated from the bottom-up at the same time. Every time $S_u \cup S_w = \oslash$, we increment the parsimony score of the tree by one. The sum of these values over all the sites is the parsimony score of the tree.

After states have been assigned to all of the internal nodes, we can obtain a labeling of them using a preorder traversal. Once again, we can compute the positions (sites) independently. The root $r$ arbitrarily assign its state to be any element of $S_r$. Next we visit the remaining nodes in turn, each time assigning a state to the node $v$ from its set $S_v$. When we visit a node $v$, we will have already set the state of its parent, $p$. If the selected state for $p$ is an element of $S_v$, then we use the same state. Otherwise we pick a state arbitrarily from $S_v$.

Fitch's algorithm requires $O(nk)$ time to compute the labeling of every node in $T$ and the optimal length (i.e., maximum parsimony score) of $T$, where $n = |S|$, and $k$ is the sequence length.

### 1.2.2 Parsimony informative sites

When computing the parsimony score, one only needs to consider *informative* sites. Parsimony-informative sites include only those sites where at least two distinct characters are observed two or more times. Consider the sequences for four taxa in Table 1.1. The four sequences can be related to one another in three different ways (see Figure 1.4). Let us now consider one site after the other in terms of the minimum changes it involves.

Site 1 has not changed and can therefore be ignored. Site 2 must have suffered at least one change, no matter how we arrange the tree. (This site could, of course, have undergone more than one change, but this assumption is not maximally parsimonious and so is not taken into consideration). For site 2, all three trees are equally likely and so the site is regarded, like site 1, as being uninformative and as such is not considered any further. Site 3 is informative since it enables use to choose the most parsimonious tree. Tree I requires one change, whereas trees II and III each requiring two changes. Finally, site 4 is also informative and identifies tree II as the most parsimonious tree for this site. Thus, trees I and II are the optimal MP trees.

So for four taxa, only three site patterns are informative: $aabb, abab, abba$, where $a$ and $b$ are two different states from $\Sigma$. Informative and non-informative sites affect

| | Sites | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| $S1$ | G | T | A | G |
| $S2$ | G | T | A | C |
| $S3$ | G | T | G | G |
| $S4$ | G | C | G | C |

**Table 1.1 DNA sequences used in Figure 1.4.**

only parsimony. Distance and likelihood methods, all sites including the constant site affect the calculation and should be included.

### 1.2.3 Exact MP search

One approach to solving MP is to evaluate all possible trees to guarantee that the most parsimonious tree is found. Thus, an algorithm is needed to generate all possible trees. One procedure for for generating all possible trees is as follows. Consider the unrooted tree $T$ for three taxa. To create the all trees with four taxa, we add the fourth taxon to each edge in $T$. Thus, the algorithm adds the $i$th taxon in a stepwise fashion to all possible trees containing the first $i - 1$ taxa until all $n$ taxa have been joined.

The above algorithm makes it clear that the number of possible trees grows by a factor that increase by two with each additional taxon. This relationship is expressed as $B(t) = \prod_{i=3}^{t}(2i - 5)$, where $B(t)$ is the number of unrooted trees for $t$ taxa. As stated earlier, for 13 taxa there are over 13 billion trees to score. Clearly, the exhaustive search method can only be used for only a relatively small number of taxa. An

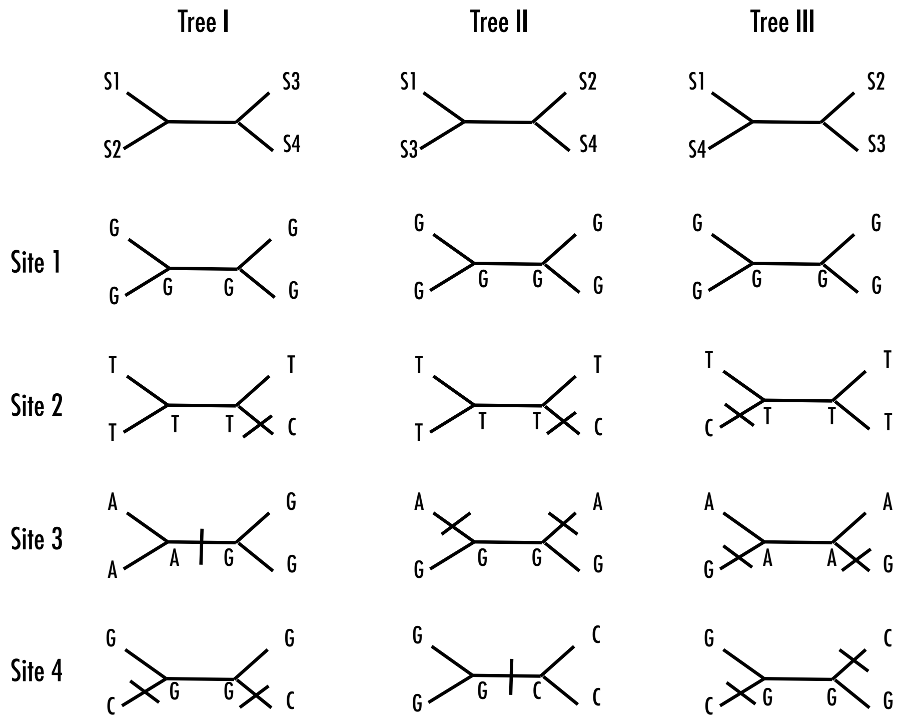


**Fig. 1.4** Finding the most parsimonious tree. Using only the informative sites (Sites 3 and 4), the optimal MP score is 3 as shown by Trees I and II. The DNA sequences represented by $S1, S2, S3$, and $S4$ are $GTAG, GTAC, GTGG$, and $GCGC$, respectively. These sequences are shown in Table 1.1.

alternative exact procedure, the branch-and-bound method [Hendy and Penny, 1982] operates implicitly by evaluating all possible trees, but cutting off paths of the search tree when it is determined that they cannot possibly lead to optimal trees. We describe a high-performance B&B algorithm in section 1.3.

### 1.2.4 MP heuristics

When data sets become too large to use the exact searching methods, one must resort to the use of heuristics. The fundamental technique is to take an initial estimate of the tree and rearrange branches in it, to reach neighboring trees. If a rearrangement yields a better scoring tree, it becomes the new ”best” tree and it is then submitted to a new round of rearrangements. The process continues until no better tree can be found in a full round.

Nearest-neighbor interchange (NNI) is one type of tree rearrangement operation (see Figure 1.5). The NNI operation effectively swaps two adjacent branches on the tree. In particular, an interior edge is removed from the tree, and the two branches connected to it at each end (i.e., a total of five branches are erased). Four subtrees remain that are disconnected from each other. These subtrees can be hooked together into a tree in three possible ways. One of the three trees is the original one, so that each nearest neighbor interchange examines two alternative trees. In an unrooted bifurcating tree with $n$ taxa, there will be $n - 3$ interior edges. At each edge, we can examine two neighboring trees. Thus, $2(n - 3)$ neighbors can be examined for a given tree. Other rearrangement operations include subtree pruning and regrafting

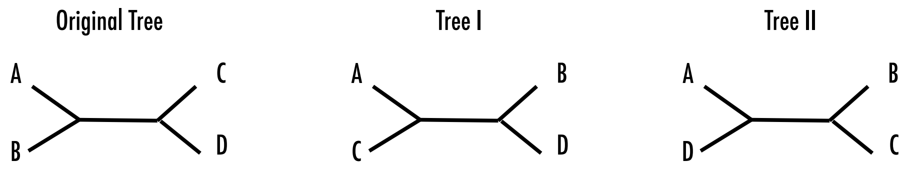


**Fig. 1.5** Nearest-neighbor interchange. An interior edge is dissolved and the four subtrees $(A, B, C, D)$ connected to it are isolated. The subtrees can be reconnected in two different ways as represented by Trees I and II.

(SPR), which breaks off part a tree and attaches it elsewhere in the tree, and tree-bisection reconnection (TBR), which breaks a phylogenetic tree into two parts and then reconnects them at a random edge.

## 1.3 EXACT MP: PARALLEL BRANCH & BOUND

Although it takes polynomial time to compute the tree cost by Fitch's method ([Fitch, 1971]), it is still very time-consuming to compute the exact MP by exhaustive search due to the enormous size of search space. Hence we use branch-and-bound (B&B) to prune the search space in phylogeny reconstruction [Bader et al., 2004, Yan and Bader, 2005]. The underlying idea of the B&B algorithm is successive decomposition of the original problem into smaller disjoint subproblems and pruning subproblems whose lower bound is greater than the upper bound until all optimal solutions are found.

### 1.3.1 Basic issues in branch and bound approach

Our B&B approach has five aspects that affect the performance of the algorithms: branching scheme, search strategy, lower bounding function, initial global upper bound, and the data structure. We will discuss these five aspects respectively in the following.

***1.3.1.1 Branching scheme*** The branch scheme decides how to decompose a subproblem in the search space. Here, each subproblem is associated with a partial tree and the objective is to find the exact MP score among those trees built from the partial tree. Our branching scheme employs the same mechanism to generate all possible unrooted binary trees for a given set of taxa. Consider the unrooted tree for three taxa. The remaining $n-3$ taxa are added to the tree in stepwise fashion as described in Section 1.2.3. Each new position location for taxon $i$ of the partial tree is considered a subproblem. Thus, a subproblem associated with the original partial tree is decomposed into a set of disjoint subproblems; each associated with a new partial tree.

***1.3.1.2 Search strategy*** The search strategy decides which of the currently open subproblems to be selected for decomposition. The two strategies most commonly used are *depth-first search (DFS)* and *best-first search (BeFS)*. DFS is space-saving strategy and BeFS is more targeted towards a better global upper bound. In the case when the initial global upper bound obtained by heuristic approaches is exactly optimal or very close to exact optimal value, there is no significant difference in the number of examined subproblems between DFS and BeFS search. Therefore DFS has more advantage for reasons of space efficiency. Since our experiment shows that heuristics can provide a very good solution, we employ DFS as our primary B&B search strategy and adopt BeFS as a secondary strategy to break ties.

***1.3.1.3 Lower bounding function of the subproblems*** Hendy and Penny ([Hendy and Penny, 1982]) use the cost of the associated partial tree as the lower bound of a subproblem. Purdom et al. used the sum of the single column discrepancy and the cost of the associated tree as the lower bound [Purdom et al., 2000]. For each column (character), the single column discrepancy is the number of states that do not occur among the taxa in the associated tree but only occur among the remaining taxa. We employ Purdom's lower bounding function since it is much tighter than the one described by Hendy and Penny.

***1.3.1.4 Initial global upper bound*** We do not compute the upper bound for each subproblem. Instead, before the B&B search, a global upper bound is obtained by a fast heuristic algorithm. We investigate the performance of both the neighbor-joining (NJ) and greedy algorithms. From experiments, we found that the tree obtained from NJ is usually much worse than the one obtained from the greedy algorithm. The greedy algorithm constructs a tree in a stepwise fashion, at each step, the new taxon is added into the best position that results in a partial tree with the minimum score. Since adding taxa in different orders yields different trees, we use the greedy algorithm with two different addition orders and use the best score as the initial global upper bound.

***1.3.1.5 Data structure*** Each subproblem in the search space of the B&B phylogeny reconstruction is associated with a partial tree and the lower bound. The lower bound serves as the priority and priority queues are used to save the open problems. Since we use DFS search, it is natural to use a priority queue for each depth. Several types of heaps can be found in literature. For simplicity, a traditional D-heap [Knuth, 1973] is chosen to represent a priority queue. A D-heap is organized as an array, using the rule that the first location is the root of the tree, and the locations $2i$ and $2i + 1$ are the children of location $i$.

### 1.3.2 Preprocessing before B&B search

We adopt a series of preprocessing in order to conduct the B&B search efficiently.

***1.3.2.1 Binary encoding of original states*** The basic operation of Fitch's method is to compute the intersection or union of state sets. Since most modern computers can perform efficient bitwise logical operations, we decide to use the binary encoding of state sets in order to implement intersection and union by bitwise AND and bitwise OR. We assign a one-to-one map between the bits of code and the character states. Given a species, if a state is present, then the corresponding bit is set to one otherwise it is set to zero.

***1.3.2.2 Decide the addition order of the species*** Our experiments show that the overall execution time of B&B phylogeny reconstruction can change drastically depending on the order in which the taxa are added. This can be explained in theory. The lower bounding function we adopt heavily depends on the cost of the associated partial tree. This can also explain why the addition order decided by max-mini rule performs best in most cases. Starting with the initial core tree of three taxa, at each step, for each of the remaining taxa, we find the best inserting position which results the minimum score. Then, we choose the taxon with maximum minimum-score to be added at its best position and go onto next step until all taxa are added. This procedure is called the max_mini approach.

***1.3.2.3 Reorder sites*** Fitch made a basic classification of sequence sites (the columns of the sequence matrix) [Fitch, 1977]. At a given site, the state that appears more than once is said to be a *non-singleton state*. A site with at most one non-singleton states is said to be a *parsimony uninformative site* since the state changes at such kind of a site can always be explained by the same number of substitutions in all topologies. At the lower levels of B&B phylogeny reconstruction, only a few sites are parsimony informative. However, with the addition of taxa, many sites turn from parsimony uninformative to parsimony informative. Hence, we may compute at which level a site turns into parsimony informative, then reorder sites so that at each level all of the parsimony informative sits are kept in a contiguous segment of memory. By reordering sites, not only is the computation on parsimony uninformative sites saved, but also the ratio of cache misses is greatly reduced.

### 1.3.3 A fast algorithm to compute tree length

In a B&B search, an enormous number of trees must be evaluated. Given an original tree, how can we compute the scores of each new tree generated by adding a taxon in the original one? As described in Section 1.2.1, Fitch's method involves one bottom-up pass and one top-down pass of the tree. Each pass computes a set of states for each internal node by different rules, the states obtained in the first pass are *preliminary states* and the states obtained in the second pass are *final states*. Goloboff proposed a method to preprocess the original tree in two passes, which takes constant time to compute the score for each new tree [Goloboff, 1993]. Gladstein described an incremental algorithm based on preliminary state sets obtained from Fitch's first pass [Gladstein, 1997]. In practice, Goloboff's method works better than Gladstein's. We developed an approach that requires preprocessing the original tree in one pass and for each new tree it takes constant time to compute the score.

Our approach is similar to Fitch's algorithm. Our first pass is identical to Fitch's first pass. However, our second pass uses the rules of Fitch's first pass—not Fitch's second pass rules. We obtain a set of states for each edge, which are the preliminary states of the root Therefore, when inserting a new taxon in the original tree at an edge, we only compare the states of the new taxon and the states of that edge. We obtain the same result as our bottom-up pass does on the new tree. If the result of the new tree is kept in memory, when we decompose this new tree later, only the top-down pass is required. Thus, in B&B search, our method saves one pass compared to Goloboff's method. Besides the B&B search, our method can also be applied to heuristic search based SPR and TBR rearrangement operations.

### 1.3.4 Parallel implementation on SMPs

To utilize the computation power of parallel computers, we implement the B&B phylogeny reconstruction on Cache-Coherent Uniform Memory Access (CC-UMA) SMPs. In the parallel implementation, each processor selects different active nodes, then processes the computation on it. We use the SPMD asynchronous model in which each processor works at its own pace and does not have to wait at predetermined points for predetermined data to become available. Since the B&B search space tends to be highly irregular, any static distribution of search space is bound to result in significant load imbalance, and the dynamic distribution methods usually involve very complex protocols to exchange subspace between processors to obtain load balance. Compared to the distributed data structure, a single shared data structure is easily maintained on SMPs since there is no load balance problem. We modify the serial data structure by adding a lock for each heap to get the shared data structure. Each heap is protected by a lock and the entire heap is locked whenever it is being modified. Due to the small size of heaps in B&B phylogeny reconstruction, the D-heap is used for simplicity and efficiency.

To minimize the concurrent access contention, a relaxed DFS search strategy is adopted. A heap is accessed if all the heaps at higher levels are empty or locked by other processors. When a processor detects that all the heaps are unlocked and empty, this processor can terminate its own execution of the algorithm.

### 1.3.5 Experimental results

We use the benchmark collection at http://www.lirmm.fr/ ranwez/PHYLO/benchmarks24.html. Each data set consists of 24 sequences and the length of each DNA sequence is 500. These tests allow comparison on trees whose internal branch lengths are not all equal, and over a wide variety of tree shapes and evolutionary rates.

We compared the running time between our serial code and PAUP* using the sub-command *bandb addseq=maxmini* on a Sun UltraSparcII workstation. Among the 20 data sets randomly chosen from the benchmark, for 10 data sets our code is 1.2-7 times faster than PAUP*, for 5 data sets our code runs as fast as PAUP*, and for 5 data sets our code is 1.2-2 times slower than PAUP*. The experiments on our parallel code was carried out on Sun E4500, a uniform-memory-access (UMA) shared-memory parallel machine with 14 UltraSparcII 400MHz processors. We conducted the experiment on 200 data sets randomly chosen from the benchmark, on average we achieve speedups of 1.92, 2.78, and 4.34, on 2, 4, and 8 processors, respectively. The above experimental results show that our strategies on the B&B phylogeny reconstruction are efficient.

## 1.4 MP HEURISTICS: DISK-COVERING METHODS

*Disk-Covering Methods (DCMs)* [Huson et al., 1999a, Huson et al., 1999b, Nakhleh et al., 2001, Roshan et al., 2004, Warnow et al., 2001] are a family of divide-and-conquer methods designed to "boost" the performance of existing phylogenetic reconstruction methods [Huson et al., 1999a, Huson et al., 1999b, Nakhleh et al., 2001, Roshan et al., 2004, Warnow et al., 2001]. All DCMs require four steps.

1. Decompose the dataset into subproblems.

2. Apply a "base" phylogenetic method to each of the subproblems.

3. Merge the subproblems.

4. Refine the resulting tree.

Variants of DCMs come from different decomposition techniques for the initial step; the last three phases are unaffected. The first DCM [Huson et al., 1999a], also called DCM1, was designed for use with distance-based methods and has provable theoretical guarantees about the sequence length required to reconstruct the true tree with high probability under Markov models of evolution [Warnow et al., 2001]. The second DCM [Huson et al., 1999b], also called DCM2, was designed to speed up heuristic searches for MP trees; we showed that when DCM2 was used with PAUP*-TBR search, it produced better trees faster on simulated datasets.

### 1.4.1 DCM3

We designed the third DCM, or *DCM3*, from the lessons learned with our first two DCMs. DCM1 can be viewed, in rough terms, as attempting to produce overlapping clusters of taxa to minimize the intracluster diameter; it produces good subproblems (small enough in size), but the structure induced by the decomposition is often poor. DCM2 computes a fixed structure (a graph separator) to overcome that drawback, but the resulting subproblems tend to be too large. Moreover, both DCM1 and DCM2 operate solely from the the matrix of estimated pairwise distances, so that they can produce only one (up to tiebreaking) decomposition. In contrast, DCM3 uses a dynamically updated *guide tree* (in practice, the current estimate of the phylogeny) to direct the decomposition—so that DCM3 will produce different decompositions for different guide trees. This feature enables us to focus the search on the best parts of the search space and is at the heart of the iterative use of the decomposition: roughly speaking, the iteration in *Rec-I-DCM3* consists of successive refinements of the guide tree. Thanks to the guide tree, DCM3 also produces smaller subproblems than DCM2: the guide tree provides the decomposition structure, but does so in a manner responsive to the phylogenetic estimation process. Finally, we designed DCM3 to be much faster than either DCM1 or DCM2 in producing the decompositions (mostly by not insisting on their optimality), since previous experiments had shown that dataset decomposition used most of the running time with DCM2.

***1.4.1.1 Short subtree graph*** An essential component of our DCM3 decomposition algorithm is computing the short subtree graph. Consider a tree $T$ on our set $S$

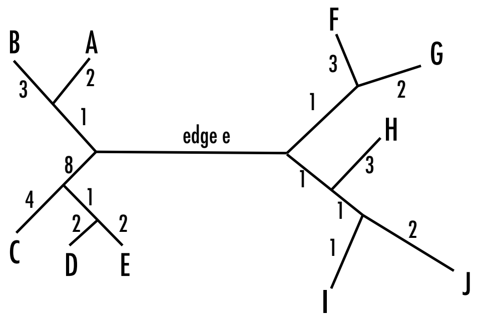


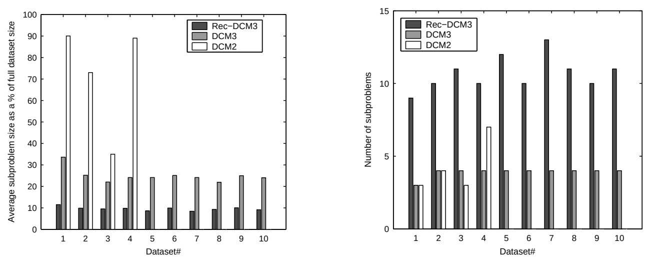
**Fig. 1.6** Short subtree around edge $e$. The leaves that form the short subtree around $e$ are $A, D, E, G,$ and $I$.

of taxa and an edge weighting $w$ of $T$, $w\colon\ E(T) \to \Re^{+}$. (A possible edge weighting is given by the Hamming distances under the MP labelling of the nodes of $T$.) We construct the *short subtree graph*, which is the union of cliques formed on "short subtrees" around each edge. Let $e$ be an internal edge (not touching a leaf) in $T$; then removing $e$ and its two endpoints from $T$ breaks $T$ into four subtrees. A *short quartet* around $e$ is composed of four leaves, one from each of these four subtrees, where each leaf is selected to be the closest (according to the edge weights) in its tree to $e$. This short quartet need not be unique: several leaves in the same subtree may lie at the same shortest distance from $e$. Thus we define the *short subtree* around $e$ to be the set $X(e)$ of all leaves that are part of a short quartet around $e$. Figure 1.6 provides an example of computing a short subtree around an edge $e$. We will use the *clique* on $X(e)$: the graph with $X(e)$ as its vertices and with every pairwise edge present, weighted according to $w$; denote this clique by $K(e)$. The *short subtree graph* is then the union, over all internal edges $e$ of the guide tree, of the $K(e)$.

***1.4.1.2 DCM3 decomposition*** Our decomposition algorithm requires finding a *centroid edge* in the guide tree $T$—that is, an edge that, when removed, produces the most balanced bipartition of the leaves. Our approach is based on the notion of a *centroid edge* in $T$—that is, an edge that, when removed, produces the most balanced bipartition of the leaves. Let $X$ be the leaves of the short subtree around a centroid edge $e$. In our experience, $X$ is always a separator of the short subtree graph, so we can define the subproblems as $A_i = X \cup C_i$, where $G - X$ has $m$ distinct connected components, $C_1, C_2, \ldots, C_m$. (Should $X$ fail to be a separator in the short subtree graph, we would then resort to computing all maximal clique separators in $G$.) Since we cannot afford to compute the short subtree graph, we cannot directly verify that $X$ is a separator. However, we can proceed without knowing the short subtree graph [Roshan et al., 2004]. By using this result, we can compute a decomposition that is not exactly that induced by the centroid edge, but that retains good characteristics (i.e., small number of small subproblems).

Finding a centroid edge $e$ through a simple tree traversal requires linear time. Computing $X(e)$ and then the subproblems $A \cup X(e)$, $B \cup X(e)$, $C \cup X(e)$, and $D \cup X(e)$ also require linear time. Thus, a DCM3 decomposition can be computed in $O(n)$ time, where $A$, $B$, $C$, and $D$ are the sets of leaves in the four subtrees obtained by deleting $e$ from $T$.

***1.4.1.3 Comparison of DCM decompositions*** We designed DCM3 in part to avoid producing large subsets, as DCM2 is prone to do. Yet, of course, the subproblems produced from a very large dataset remain too large for immediate solution by a base method. Hence we used the approach successfully pioneered by Tang and Moret with DCM-GRAPPA [Tang and Moret, 2003] and used DCM3 recursively, producing smaller and smaller subproblems until every subproblem was small enough to be solved directly. Figure 1.7 shows that DCM3 produces subproblems much smaller than those produced by DCM2. (Rec-DCM3 in this series of tests was set up to recurse until each subproblem was of size at most one eighth of the original size.)



(a) mean relative subproblem size (b) number of subproblems

**Fig. 1.7** Comparison of DCM2, DCM3 and Recursive-DCM3 decompositions. DCM2 decompositions on datasets 5–10 could not be computed due to memory limitations.

***1.4.1.4 Subtree construction and assembly*** Once the dataset is decomposed into overlapping subsets $A_1, A_2, \ldots, A_m$ (for us, $m \leq 4$ is typical), subtrees are constructed for each subset, $A_i$, using the chosen "base method," and then combined using the Strict Consensus Merger [Huson et al., 1999a, Huson et al., 1999b] to produce a tree on the combined dataset. The proof that the resulting tree is accurate (i.e., agrees, with high probability and in the limit, with the unknown underlying "true tree") is similar to the argument shown in [Huson et al., 1999a]).

Our *Rec-I-DCM3* algorithm takes as input the set $S = \{s_1, \ldots, s_n\}$ of $n$ aligned biomolecular sequences, the chosen base method, and a starting tree $T$. In our experiments, we have used TNT (with default settings) as our base method, since it is the hardest to improve (in comparison, the PAUP* implementation of the parsimony ratchet [Bininda-Emonds, 2003] is easier to improve). Our algorithm produces smaller subproblems by recursively applying the centroid-edge decomposition until each subproblem is of size at most $k$. The subtrees are then computed, merged, and resolved (from the bottom-up, using random resolution) to obtain a binary tree on the full dataset. These steps are repeated for a specified number of iterations.

### 1.4.2 Experimental design

The experimental evaluation of algorithms for phylogenetic reconstruction is a difficult endeavor (see [Moret, 2002, Moret and Warnow, 2002] for details). Because credible simulations of evolution remain lacking at the scale of 10,000 or more taxa, we chose to use biological datasets in our study. This choice ensures biological relevance of our results, but it prevents us from evaluating the accuracy of reconstructed trees, since the “true” tree is not available. However, other work from our group [Williams et al., 2004] tells us that we need to achieve excellent approximation of the parsimony score (tree length) in order to have any chance at reconstructing the true topology. Thus, we focused our testing on the quality of approximation in terms of the parsimony score.

***1.4.2.1 Biological Datasets:*** We present our results on the ten large biological datasets—because of their large size, all but one are RNA data, ranging from a smallest set of 1,322 sequences to a largest of 13,921 sequences, all with sequence lengths between 800 and 1,600. Seven of these ten sets have over 4,500 sequences and thus are not, in practice, accurately analyzable with existing MP heuristics.

1. A set of 1,322 aligned large subunit ribosomal RNA of all organisms (1,078 sites) [Wuyts et al., 2002].

2. A set of 2,000 aligned Eukaryotes ribosomal RNA sequences (1,326 sites) obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.

3. A set of 2,594 *rbcL* DNA sequences (1,428 sites) [Kallerjo et al., 1998].

4. A set of 4,583 aligned 16s ribosomal Actinobacteria RNA sequences (1,263 sites) [Maidak et al., 2000].

5. A set of 6,590 aligned small subunit ribosomal Eukaryotes RNA sequences (1,661 sites) [Wuyts et al., 2002].

6. A set of 7,180 aligned ribosomal RNA sequences (1,122 sites) from three phylogenetic domains obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.

7. A set of 7,233 aligned 16s ribosomal Firmicutes (bacteria) RNA sequences (1,352 sites) [Maidak et al., 2000].

8. A set of 8,506 aligned ribosomal RNA sequences (851 sites) from three phylogenetic domains, plus organelles (mitochondria and chloroplast), obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.

9. A set of 11,361 aligned small subunit ribosomal Bacteria RNA sequences (1,360 sites) [Wuyts et al., 2002].

10. A set of 13,921 aligned 16s ribosomal Proteobacteria RNA sequences (1,359 sites) [Maidak et al., 2000].

***1.4.2.2 Parameters and measurements:*** We chose to test performance during the first 24 hours of computation on each dataset for each method, taking hourly "snapshots" along the way in order to evaluate the progress of each method. We asked the following two questions: (i) how much of an improvement is gained by using *Rec-I-DCM3* versus TNT, if any? and (ii) how long does the best TNT trial (out of five runs) take to attain the average MP score obtained at 24 hours by *Rec-I-DCM3*? To answer these questions, we ran TNT and *Rec-I-DCM3(TNT)*, which uses TNT as its base method, on our ten biological datasets, using five independent runs, all on the same platform, with computed variances for all measurements.

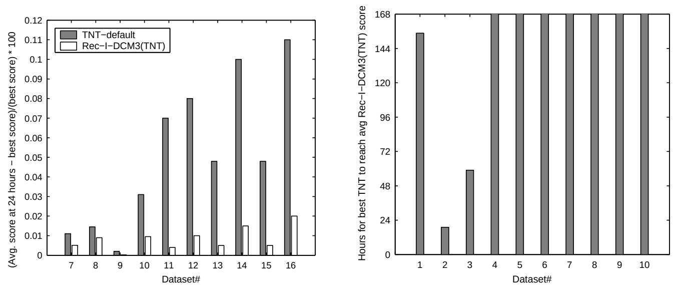
***1.4.2.3 Implementation and platform:*** Our DCM implementations are a combination of LEDA, C++, and Perl scripts. The TNT Linux executable was obtained from Pablo Goloboff, one of the authors of TNT. We ran our experiments on three sets of processors, all running Linux: the `Phylofarm` cluster of 9 dual 500MHz

Pentium III processors; a part of the 132-processor `SCOUT` cluster, consisting of 16 dual 733MHz Pentium III processors, and the `Phylocluster` of 24 dual 1.5GHz AMD Athlon processors, all at the University of Texas at Austin. For each dataset all the methods were executed on the same cluster; larger datasets were run on the faster machines.

### 1.4.3 Results

We defined the "optimal" MP score on each dataset to be the best score found over all five runs among all methods in the 24-hour period we allowed; on our datasets, this optimal score was always obtained by *Rec-I-DCM3(TNT)*. On each dataset and for each method, we computed the average MP score at hourly intervals and reported this value as a percentage of deviation from optimality. In our experiments, on every dataset and at every point in time (within these 24 hours), the best performance was obtained by *Rec-I-DCM3(TNT)*. Since only error rates less than 0.01% are tolerable, *Rec-I-DCM3*'s performance is very impressive; all trees are at least 99.99% correct. TNT, on the other hand, failed to reach this level of accuracy consistently—especially on datasets with more than 4,500 sequences.

Figure 1.8(a) shows the performance of *Rec-I-DCM3(TNT)* and of TNT at 24 hours. As the size of the dataset increases, the relative error in MP scores increases, but at a much faster rate for TNT than for *Rec-I-DCM3(TNT)*, so that the accuracy gap between the two increases quite rapidly. Figure 1.8(b) indicates how long it took TNT, in the best of five runs, to match the average scores obtained by *Rec-I-*

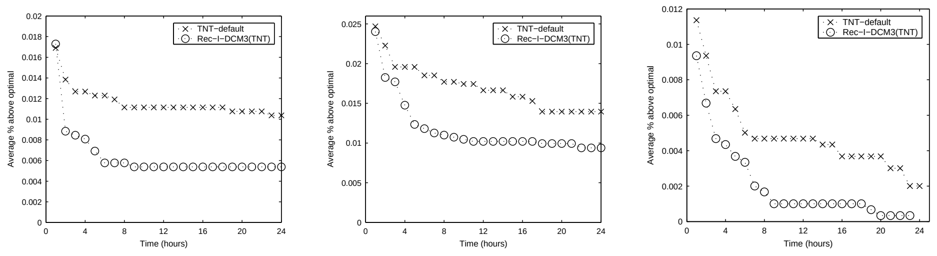


*(a) deviation from optimal* *(b) TNT time to optimal*

**Fig. 1.8** Part (a) shows the average deviation above optimal after 24 hours by TNT and *Rec-I-DCM3(TNT)*; Part (b) shows the time taken by the single best TNT trial, extended to run for up to a week, to match the average *Rec-I-DCM3(TNT)* score at 24 hours—bars that reach the top indicate that TNT could not reach a match after a week of computation.

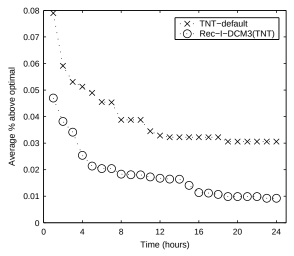
*DCM3(TNT)* after 24 hours—we stopped the clock after one week of computation if the TNT run had not achieved a match by then, something that happened on the seven largest datasets. (The standard deviations of the MP scores at 24 hours for all the methods on all the datasets were very low, at most 0.035%.)

Figure 1.9 compares the time-dependent behaviors of TNT and *Rec-I-DCM3(TNT)*

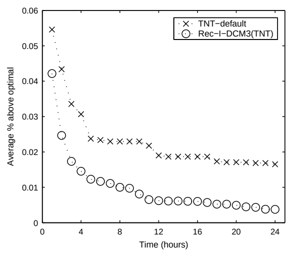


*(a) Dataset #1* *(b) Dataset #2* *(c) Dataset #3*

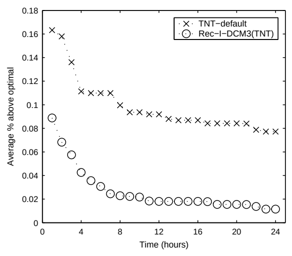
**Fig. 1.9** Average MP scores of TNT and *Rec-I-DCM3(TNT)* on datasets 1, 2, and 3, given as the percentage above the optimal score. Note: the vertical range varies across the datasets.



(a) Dataset #4

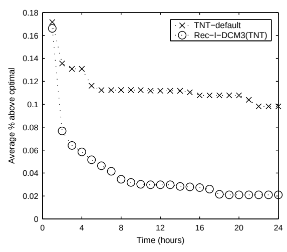


(b) Dataset #5

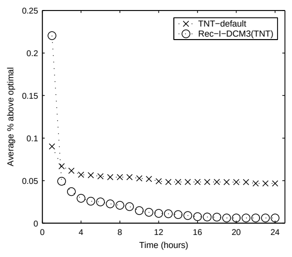


(c) Dataset #6

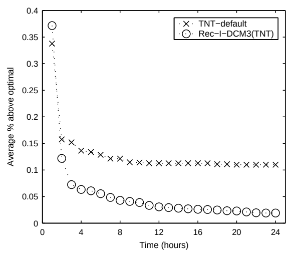
**Fig. 1.10** Average MP scores of TNT and *Rec-I-DCM3(TNT)* on datasets 4, 5, and 6, given as the percentage above the optimal score. Note: the vertical range varies across the datasets.



(a) Dataset #8



(b) Dataset #9



(c) Dataset #10

**Fig. 1.11** Average MP scores of TNT and *Rec-I-DCM3(TNT)* on datasets 8, 9, and 10, given as the percentage above the optimal score. Note: the vertical range varies across the datasets.

on our three smallest datasets (1, 2, and 3), while Figure 1.10 shows the same for three medium datasets (4, 5, and 6). and Figure 1.11 shows the same for our three largest datasets (8, 9, and 10). (It should be noted that the 24-hour time limit was perhaps overly limiting for the largest dataset: a quick look at the curves appears to indicate that even *Rec-I-DCM3(TNT)* has not yet reached a plateau at that point.) The improvement achieved by boosting TNT with *Rec-I-DCM3* is significant on all datasets as well as at all time intervals. In particular, note that the boosted version of

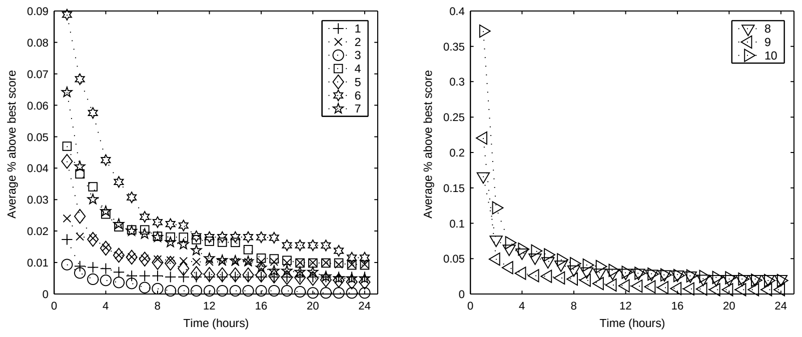


**Fig. 1.12** Decrease in error rates with time on all datasets for *Rec-I-DCM3(TNT)*

TNT shows much stronger decreases in MP scores in the first several hours than the unboosted version.

Figure 1.12 shows how the error rate (deviation above the optimal MP score) of *Rec-I-DCM3(TNT)* decreases with computation time on each of the ten datasets. While the initial trees computed for the large datasets tend to exhibit large error (as large as 0.35%), the error drops very rapidly—even more rapidly for the large datasets than for the smaller ones. Thus, not only do the error rates of *Rec-I-DCM3(TNT)* fall more rapidly than those of TNT alone, but they have a positive second derivative: the larger they are, the faster they fall.

## 1.5 SUMMARY AND OPEN PROBLEMS

Maximum parsimony is the most popular optimization criterion for analyzing large datasets. Because of the importance of MP analyses in phylogeny reconstruction, systematists and algorithm researchers have studied existing methods (specifically implementations of heuristics in different software packages) to see which performed the best. We have provided an overview of two high-performance phylogenetic algorithms, which dramatically outperform traditional techniques. Consequently, larger analyses can be performed within a reasonable time period. Each performance gain brings us closer to reconstructing the “Tree of Life”. Yet, much work still remains to be done! Below, we provide a biased sample of open problems that appear to be the most promising or important avenues for further exploration.

- Tree accuracy. Bootstrapping is one technique for estimating the support of the inferred tree [Felsenstein, 2003]. The bootstrap proceeds by resampling the original data matrix with replacement of the sites. The process is repeated many times (1,000 times or more) and phylogenies are reconstructed for each bootstrap replicate. The bootstrap support for any internal edge is the number of times it was recovered during the bootstrapping procedure. However, it is not clear how to access the accuracy of large-scale phylogenies on real datasets as a single analysis may require weeks or months to complete.

- Evolutionary models for DNA sequences. Of course, no simulation can be accurate enough to replace real data. However, simulated datasets enable evaluations of solution quality (because the model, and thus the “true” answer, is known) and can be generated in arbitrarily large numbers to ensure statistical significance. Often times, the performance ranking of phylogenetic methods on simulated data is different from that on real data. Hence, realistic models for generating simulated data are needed.

- Stopping criteria. Currently, phylogenetic analyses are stopped when one is tired of waiting for a result. Depending on the user's current needs, an analysis can be short (i.e., 24 hours) or it could run for several months. However, there is little work on determining when a search should terminate.

- Multiple sequence alignment. Although alignment was mentioned briefly in Section 1.1.1, it is probably the most crucial phase of inferring an accurate tree. More work is needed to understand the effect sequence alignment has on inferring the "true" evolutionary tree.

**Acknowledgments**

This work was supported by the National Science Foundation under grants ACI-00-81404 (Bader), ACI-00-93039 (Bader), ANI 02-03584 (Moret), DEB 01-20709 (Moret), DEB 99-10123 (Bader), EF 03-31654 (Bader and Moret), IIS 01-13095 (Moret), IIS 01-21377 (Moret), by an Alfred P. Sloan Foundation Postdoctoral Fellowship in Computational Molecular Biology DE-FG03-02ER63426 (Williams), and by the Radcliffe Institute for Advanced Study (Williams), We thank Pablo Goloboff for providing us with a Linux executable for TNT.

## REFERENCES

Bader et al., 2001. Bader, D., Moret, B. M., and Vawter, L. (2001). Industrial applications of high-performance computing for phylogeny reconstruction. In Siegel, H., editor, *Proceedings of SPIE Commercial Applications for High-Performance Computing*, volume 4528, pages 159–168, Denver, CO. SPIE.

Bader et al., 2004. Bader, D. A., Hart, W. E., and Phillips, C. A. (2004). Parallel algorithm design for branch and bound. In Greenberg, H., editor, *Tutorials on Methodologies and Applications in Operation Research*, chapter 5, pages 1–44. Academic Press.

Bininda-Emonds, 2003. Bininda-Emonds, O. (2003). Ratchet implementation in PAUP*4.0b10. available from http://www.tierzucht.tum.de:8080/WWW/Homepages/Bininda-Emonds.

Bruno et al., 2000. Bruno, W., Socci, N., and Halpern, A. (2000). Weighted neighbor joining: A likelihood-based approach to distance-based phylogeny reconstruction. *Mol. Biol. Evol.*, 17(1):189–197.

Cracraft, 2002. Cracraft, J. (2002). The seven great questions of systematic biology: An essential foundation for conservation and the sustainable use of biodiversity. *Ann. Missouri Bot. Gard.*, 89:127–144.

Felsenstein, 2003. Felsenstein, J. (2003). *Inferring Phylogenies*. Sinauer Associates.

Felsenstein, 89. Felsenstein, J. (89). Phylogenetic inference package (PHYLIP), version 3.2. *Cladistics*, 5:164–166.

Fitch, 1977. Fitch, W. (1977). On the problem of discovering the most parsimonious tree. *The American Naturalist*, 111(978):223–257.

Fitch, 1971. Fitch, W. M. (1971). Toward defining the course of evolution: minimal change for a specific tree topology. *Syst. Zool.*, 20:406–416.

Foulds and Graham, 1982. Foulds, L. R. and Graham, R. L. (1982). The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49.

Gascuel, 1997. Gascuel, O. (1997). BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Mol. Biol. Evol.*, 14:685–695.

Gladstein, 1997. Gladstein, D. S. (1997). Efficient incremental character optimization. *Cladistics*, 13:21–26.

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

Goloboff, 1993. Goloboff, P. A. (1993). Character optimization and calculation of tree lengths. *Cladistics*, 9:433–436.

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

Hendy and Penny, 1982. Hendy, M. and Penny, D. (1982). Branch and bound algorithms to determine minimal evolutionary trees. *Math. Biosci.*, 59:277–290.

Huelsenbeck and Ronquist, 2001. Huelsenbeck, J. P. and Ronquist, F. (2001). MR-BAYES: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17(8):754–755.

Huson et al., 1999a. Huson, D., Nettles, S., and Warnow, T. (1999a). Disk-covering, a fast-converging method for phylogenetic tree reconstruction. *Journal of Computational Biology*, 6:369–386.

Huson et al., 1999b. Huson, D., Vawter, L., and Warnow, T. (1999b). Solving large scale phylogenetic problems using DCM2. In *Proc. 7th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB'99)*, pages 118–129. AAAI Press.

Jukes, 1969. Jukes, T. (1969). Evolution of protein molecules. In Munro, M., editor, *Mammalian Protein Metabolism*, volume III, pages 21–132. Academic Press, New York.

Kallerjo et al., 1998. Kallerjo, M., Farris, J. S., Chase, M. W., Bremer, B., and Fay, M. F. (1998). Simultaneous parsimony jackknife analysis of 2538 *rbcL* DNA sequences reveals support for major clades of green plants, land plants, seed plants, and flowering plants. *Plant. Syst. Evol.*, 213:259–287.

Kimura, 1980. Kimura, M. (1980). A simple model for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution*, 16:111–120.

Knuth, 1973. Knuth, D. (1973). *The Art of Computer Programming: Sorting and Searching*, volume 3. Addison-Wesley Publishing Company, Reading, MA.

Maidak et al., 2000. Maidak, B., Cole, J., Lilburn, T., Jr, C. P., Saxman, P. R., Stredwick, J., Garrity, G., Li, B., Olsen, G., amanik, S. P., Schmidt, T., and Tiedje,

J. (2000). The RDP (ribosomal database project) continues. *Nucleic Acids Research*, 28:173–174.

Moret, 2002. Moret, B. (2002). Towards a discipline of experimental algorithmics. In Goldwasser, M., Johnson, D., and McGeoch, C., editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, volume 59 of *DIMACS Monographs*. American Mathematical Society.

Moret et al., 2002. Moret, B., Roshan, U., and Warnow, T. (2002). Sequence length requirements for phylogenetic methods. In *Proc. 2nd Int'l Workshop Algorithms in Bioinformatics (WABI'02)*, volume 2452 of *Lecture Notes in Computer Science*, pages 343–356. Springer-Verlag.

Moret et al., 2005. Moret, B., Tang, J., and Warnow, T. (2005). Reconstructing phylogenies from gene-content and gene-order data. In Gascuel, O., editor, *Mathematics of Evolution and Phylogeny*, pages 321–352. Oxford University Press.

Moret and Warnow, 2002. Moret, B. and Warnow, T. (2002). Reconstructing optimal phylogenetic trees: A challenge in experimental algorithmics. In Fleischer, R., Moret, B., and Schmidt, E., editors, *Experimental Algorithmics*, volume 2547 of *Lecture Notes in Computer Science*, pages 163–180. Springer-Verlag.

Moret and Warnow, 2005. Moret, B. and Warnow, T. (2005). Advances in phylogeny reconstruction from gene order and content data. In Zimmer, E. and Roalson, E., editors, *Methods in Enzymology: Producing the Biochemical Data II*. Elsevier. to appear.

Nakhleh et al., 2002. Nakhleh, L., Moret, B., Roshan, U., John, K. S., and now, T. W. (2002). The accuracy of fast phylogenetic methods for large datasets. In *Proc. 7th Pacific Symp. Biocomputing (PSB'2002)*, pages 211–222. World Scientific Pub.

Nakhleh et al., 2001. Nakhleh, L., Roshan, U., St. John, K., Sun, J., and Warnow, T. (2001). Designing fast converging phylogenetic methods. In *Proc. 9th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB'01)*, volume 17 of *Bioinformatics*, pages S190–S198. Oxford Univeristy Press.

Olsen et al., 1994. Olsen, G., Matsuda, H., Hagstrom, R., and Overbeek, R. (1994). fastdnaml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Comput. Appl. Biosci.*, 10:41–48.

Purdom et al., 2000. Purdom, P. J., Bradford, P., Tamura, K., and Kumar, S. (2000). Single column discrepancy and dynamic max-mini optimization for quickly finding the most parsimonious evolutionary trees. *Bioinfomatics*, 2(16):140–151.

Roshan et al., 2004. Roshan, U., Moret, B. M. E., Williams, T. L., and Warnow, T. (2004). Rec-I-DCM3: a fast algorithmic technique for reconstructing large phylogenetic trees. In *Proc. IEEE Computer Society Bioinformatics Conference (CSB 2004)*, pages 98–109. IEEE Press.

Saitou and Nei, 1987. Saitou, N. and Nei, M. (1987). The neighbor-joining method: A new method for reconstructiong phylogenetic trees. *Mol. Biol. Evol.*, 4(406–425).

Sanderson et al., 1993. Sanderson, M., Baldwin, B., Bharathan, G., Campbell, C., Ferguson, D., Porter, J., Dohlen, C. V., Wojciechowski, M., and Donoghue, M. (1993). The growth of phylogenetic information and the need for a phylogenetic database. *Systematic Biology*, 42:562–568.

Steel, 1994. Steel, M. (1994). The maximum likelihood point for a phylogenetic tree is not unique. *Systematic Biology*, 43(4):560–564.

Swofford, 2002. Swofford, D. L. (2002). PAUP*: Phylogenetic analysis using parsimony (and other methods). Sinauer Associates, Underland, Massachusetts, Version 4.0.

Tang and Moret, 2003. Tang, J. and Moret, B. (2003). Scaling up accurate phylogenetic reconstruction from gene-order data. In *Proc. 11th Int'l Conf. on Intelligent Systems for Molecular Biology ISMB'03*, volume 19 (Suppl. 1) of *Bioinformatics*, pages i305–i312.

Warnow et al., 2001. Warnow, T., Moret, B., and St. John, K. (2001). Absolute convergence: True trees from short sequences. In *Proc. 12th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'01)*, pages 186–195. SIAM Press.

Williams et al., 2004. Williams, T., T. Berger-Wolf, B. M., Roshan, U., and Warnow, T. (2004). The relationship between maximum parsimony scores and phylogenetic tree topologies. Technical Report TR-CS-2004-04, Department of Computer Science, The University of New Mexico.

Wuyts et al., 2002. Wuyts, J., de Peer, Y. V., Winkelmans, T., and Wachter, R. D. (2002). The European database on small subunit ribosomal RNA. *Nucleic Acids Research*, 30:183–185.

Yan and Bader, 2005. Yan, M. and Bader, D. A. (2005). High performance algorithms for phylogeny reconstruction with maximum parsimony. In Aluru, S., editor, *Handbook on Computational Molecular Biology*. CRC Press. to appear.