1

Disk Covering Methods: improving the accuracy and speed of large-scale phylogenetic analyses

Tandy Warnow
The University of Texas at Austin

1.1 Introduction

Large-scale phylogeny reconstruction poses several challenges to the algorithms designer. To begin with, the fundamental objective in phylogeny reconstruction is to reconstruct, as accurately as possible, the tree that produced the input dataset (typically aligned biomolecular sequences), rather than to solve any particular numeric optimization problem. Therefore, all reconstruction methods are studied (typically in simulation - see [16, 18, 19] for some examples of simulation studies) with respect to what is called their "topological accuracy" - whereby the reconstructed trees are compared against the model tree and the differences

0-8493-8597-0/01/$0.00+$1.50

between the branching order in the two trees are quantified. Some reconstruction methods operate in polynomial time and have been shown to perform well with respect to the topological accuracy of the reconstructed trees under many model conditions; however, recent work has shown that for large model trees with high interleaf distances, popular polynomial time methods do not produce trees with acceptable levels of topological accuracy [1, 28, 30, 31, 32]. For this reason, among others, most systematists prefer methods that attempt to solve either maximum parsimony (MP) or maximum likelihood (ML), two very hard optimization problems (MP provably NP-hard [13], and ML harder in practice than MP). While many heuristics exist for these optimization problems, it is not clear that these heuristics are able to obtain good enough solutions (for their criteria) on large datasets - that is, they do not scale well. Thus, large-scale phylogenetic analysis is a difficult challenge. (See [17, 23] for an introduction to phylogeny reconstruction).

This chapter will present a meta-technique for large-scale phylogenetic analysis which has been shown to improve both types of phylogenetic reconstruction methods. The basic meta-technique first "decomposes" the dataset into overlapping subsets. Trees are then constructed on each of the resultant datasets using a favored phylogenetic reconstruction method; since the subsets overlap, these trees also overlap on their leaf sets. These subtrees can then be merged together, using a preferred "supertree" method, into a tree on the full set of taxa. We call the class of methods using this basic structure "Disk-Covering Methods", or DCMs, for historic reasons. Our research shows that DCMs can make phylogenetic reconstruction more accurate and/or faster, depending upon the particular vulnerabilities of the base method. Thus, a DCM is designed to "boost" the performance of a given "base method". Our current DCMs have focused on two different types of base methods – polynomial time distance-based methods, such as neighbor joining [37], and heuristics for maximum parsimony implemented in software such as PAUP* [42] and TNT (see [14] for a description of TNT). These DCMs differ in their design, because of the particular aspects of the base method. Most noticeably, they use different decomposition strategies. The DCM we use for neighbor joining uses a decomposition that produces very small subproblems with small evolutionary diameters (i.e., interleaf-distances), while the decomposition strategy used in the DCM for maximum parsimony produces larger subproblems, with a maximum subproblem size that is still reasonably large. In addition, the DCM for maximum parsimony is used iteratively within a heuristic search in order to obtain improved results. However, we do not expect our current approaches for improving heuristic maximum parsimony or neighbor joining to be the best that can be achieved, and so new DCMs will continue to be useful for these problems. Furthermore, new DCMs will likely be needed as new problems (such as maximum likelihood or phylogenetic multiple sequence alignment [38]) are considered. Thus, we believe that this type of algorithm design, in which base methods are used on carefully selected subproblems and solutions then merged together, represents a promising approach to phylogeny reconstruction, but that DCMs are really in their infancy. We write this chapter, therefore, in the hope that by presenting both the intuition behind the design strategies and the details involved in designing DCMs for specific base methods, algorithms researchers will be able to design their own DCMs as new challenges arise.

The structure of the chapter is as follows. In Section 1.2 we present the basic issues involved in phylogenetic analysis, including definitions of the basic optimization problems, stochastic models of evolution, statistical aspects of a phylogenetic estimation, and algorithmic issues. In Section 1.3 we present the basic divide-and-conquer strategies we use in our various DCMs. These techniques rely heavily on the theory of triangulated graphs, which we present in Section 1.4. We then discuss general design issues for DCMs in Section 1.5. We then begin our description of a few specific DCMs in order to illustrate these techniques. We begin in Section 1.6 with the DCM we designed for use with maximum parsimony heuristics. We continue in Section 1.7 with a description of DCMs that were designed for use with distance-based methods like neighbor joining. Finally, we conclude in Section 1.8 with a discussion of further research directions, and some specific open problems.

1.2 Phylogenetic analysis

1.2.1 Stochastic models of sequence evolution.

Stochastic models have been proposed for all types of biomolecular sequences, including RNA, DNA, and amino-acid. Of these, DNA have the simplest models, since typically these models do not include any structural constraints, as would be generally true in RNA or amino-acid models. Hence, we will focus here on describing the models that have been proposed for DNA evolution.

A model of DNA sequence evolution must describe the probability distribution of the four states, A, C, T, G, at the root, the evolution of a random site (i.e., position within the DNA sequence) and how the evolution differs across the sites. Typically the probability distribution at the root is uniform (so that all sequences of a fixed length are equally likely). The evolution of a single site on a given edge e is modeled through a collection of parameters - one is the "length" $l(e)$ of the edge, which determines the expected number of changes of a random site on that edge, and the other is a substitution probability matrix, which determines the probabilities of each substitution of one nucleotide by another in a single substitution. These together can be fully expressed by a single "stochastic substitution matrix," $M(e)$, which for DNA is a 4×4 matrix in which every row sums to 1; because this matrix allows for more than one change, the diagonal entries can be non-zero. Note that the matrix $M(e)$ can have up to 12 free parameters. The simplest such model is the Jukes-Cantor model, with one free parameter, and the most complex is the General Markov model, with all 12 parameters [40].

DEFINITION 1.1 The General Markov (GM) model of single-site evolution is defined as follows.

1. The nucleotide in a random site at the root is drawn from a known distribution, in which each nucleotide has positive probability.
2. The probability of each site substitution on an edge e of the tree is given by a 4×4 stochastic substitution matrix $M(e)$ in which $det(M(e))$ is not 0, 1, or -1.

Note that these models only describe the evolution of a single site down the tree. To model how a sequence evolves we would also need to describe how the different sites evolve. Almost all models of sequence evolution assume that different sites evolve independently (a notable exception is the covarion model [43]), but most phylogenetic analyses are based upon models for which the independence of different sites is assumed. The majority of models used in inference allow for sites to evolve differently, but almost all assume that the differences between sites is limited in the following way. The assumption of how sites differ is expressed by saying that for every site i we have a rate r_i, so that the expected number of changes on the edge e is simply the product of the edge length $l(e)$ with the rate r_i. If all sites evolve under the same rate, then $r_i = r_j$ for all i, j. These $\{r_i\}$ are the "rates-across-sites", with r_i the rate for the i^{th} site. Typically these rates are drawn from some distribution. Note that what this expresses is that if a site i is expected to evolve twice as fast as site j on one edge, then it is expected to evolve twice as fast on every edge – that is, sites speed up or slow down identically under all conditions. While this is not necessarily a reasonable assumption in molecular evolution, it is the underlying assumption of models in practice.

In this chapter, we use the GM model with the assumption that $r_i = r_j$, so as to simplify the analysis.

We denote a model tree in the GM model as a pair, $(T, \{M_e : e \in E(T)\})$, or more simply as (T, M). We assume that the number of changes of a given site on a given edge obeys a Poisson distribution. For each edge $e \in E(T)$, we define the length of the edge $\lambda(e)$ to be $-\log|det(M_e)|$. This allows us to define the matrix of leaf-to-leaf distances, $[\lambda_{ij}]$, with $\lambda_{ij} = \sum_{e \in P_{ij}} \lambda(e)$ and where P_{ij} is the path in T between leaves i and j. Note that $[\lambda_{ij}]$ is a symmetric matrix. It is a well-known fact that, given the distance matrix $[\lambda_{ij}]$, it is easy to recover the underlying leaf-labelled tree T in polynomial time.

This general model of site evolution subsumes the great majority of other models examined in the phylogenetic literature, including the Hasegawa-Kishino-Yano (HKY) model, the Kimura 2-parameter model (K2P), the Kimura 3-ST model (K3ST), the Jukes-Cantor model (JC), etc. These models are all special cases of the General Markov model, because they place restrictions on the form of the stochastic substitution matrices (see [26] for more information about stochastic models of evolution).

1.2.2 Statistical performance issues.

Once a stochastic model of evolution is stated, it becomes possible to discuss statistical inference under the model, define and develop explicitly statistical estimation methods (such as maximum likelihood), and to ask about the performance of phylogeny reconstruction methods under the model. One of the aspects of performance that is typically considered is whether a reconstruction method is "statistically consistent" under the given model. Put simply, a reconstruction method is statistically consistent under a given model of evolution, if, for all model trees under the model, as the sequence length increases, the probability of reconstructing the model tree goes to 1. This is a mathematical question and establishing statistical consistency requires a mathematical proof. Such proofs have been obtained for many methods, including most distance-based methods (including neighbor-joining [37]) and maximum likelihood (which seeks the model tree that maximizes the probability of the data) under the GM model, and hence under all its submodels.

Another aspect of statistical performance under a model is its "convergence rate", which roughly speaking asks how quickly the error rate in the estimation goes to 0 as a function of the sequence length. In order to make this statement precise, we must provide a definition of topological error. While there are many ways to quantify this, the one that is used most typically in the phylogenetics research community is the Robinson-Foulds [33] rate. This is given as follows.

DEFINITION 1.2 Let T be the true tree, and let T' be an estimated tree, both on the same set of n leaves. For each edge e in T, there is an associated bipartition $\pi(e)$ defined on the leaf set of T which is produced by deleting the edge e from T. We can therefore identify T with its set $C(T) = \{\pi(e) : e \in E(T)\}$. Similarly we can identify the tree T' by its set $C(T')$, defined in the analogous way. The **Robinson-Foulds** distance between T and T' is then the average of $|C(T) - C(T')|/(n-3)$ and $|C(T') - C(T)|/(n-3)$; the first of these two values is called the *missing edge* or *false negative* rate, and the second of these two values is called the *false positive* rate.

These are *rates* because they are divided by $n-3$, where n is the number of leaves in each tree ($n-3$ corresponds to the number of internal edges in a binary tree on n leaves). Error rates that are below 5% are desired, and above 10% are unacceptable - unless the data are just too poor to allow for greater resolution.

Two methods which are both statistically consistent under a model may have very different convergence rates, with one method producing trees with much lower error rates than the other, at most "reasonable" sequence lengths. Thus, the convergence rate of a method is highly significant when it comes to predicting its performance on different datasets.

While statistical consistency is relatively easy to establish (the proofs are not difficult generally), mathematical analyses of the convergence rates of different methods is very difficult (see [1, 9, 10] for some initial results). For this reason, the performance of phylogeny reconstruction methods is typically evaluated in simulation.

1.2.3 Maximum parsimony.

We now define the maximum parsimony problem.

- Input: Set S of sequences, each over an alphabet A, and of the same length k.
- Output: a tree T with leaves labelled by S and with internal nodes labelled by other elements of A^k, so as to minimize the total "length" of the tree, which is defined to be $\sum_e H(e)$, where $H(e)$ denotes the Hamming distance between the sequences labelling the different endpoints of e (i.e., the number of sites at which the sequences differ).

Finding the optimal trees under maximum parsimony (MP) is an NP-hard problem, and so hard to solve exactly for datasets beyond about 30 or so taxa (if the tree T is fixed, then the problem of assigning sequences to the internal nodes so as to minimize the total length of the tree is easily solved using dynamic programming in polynomial time [12]). Because MP is important in practice, many heuristics exist which attempt to solve the problem through a combination of hill-climbing and randomization to get out of local optima. These heuristics may actually find optimal solutions, though current approaches do not provide sufficiently good lower bounds to make it possible to assess the degree of suboptimality in a given analysis. Current practice therefore tends to involve running a favored heuristic until it seems that better solutions will not be found. Such analyses can take a few days on moderate sized datasets, to weeks or months on large datasets.

MP is not statistically consistent under the GM model, which means that there are some model trees so that as the sequence length increases, the probability that an exact solution to MP would yield the true tree does *not* provably approach 1 [11]. Even so, MP is a popular method, and many heuristics exist to attempt to solve MP. (For more on MP, see [17].)

1.2.4 Distance-based methods.

Distance-based methods operate in two phases: first they use statistically-based techniques to estimate pairwise distances, and then they construct a tree on the basis of these estimated distances. Provided that the appropriate technique is used to estimate pairwise distances, and a good method is used in the second step to construct a tree from the distances, the combined two-phase reconstruction is statistically consistent. The most popular distance-based methods are polynomial-time, and hence these methods are quite attractive.

Of the various distance-based methods, neighbor-joining is probably the most popular; it is statistically consistent under the GM model and performs well in many simulation studies by comparison to other distance-based methods. On the other hand, the only mathematical theory about its convergence rate shows that it can require sequence lengths to be exponential in the maximum leaf-to-leaf distance within the model tree, in order to produce the topologically correct true tree with high probability (see below). Since sequence lengths are generally not extremely long, this is a vulnerability of neighbor joining with respect to large-scale phylogeny reconstruction. (This theoretical statement also holds true for other distance-based methods and so this vulnerability of neighbor joining is not unique.) Simulation studies have verified that neighbor joining's performance degrades as the interleaf-distances increase without a corresponding increase in the sequence length [37], and so from an empirical standpoint this vulnerability seems to be significant. (It is worth noting that all methods seem to degrade in performance with increasing interleaf-distances, with the degradation of neighbor-joining less so than that of some other distance-based methods, but worse - it seems - than some sequence-based methods.)

The basic theorem for the convergence rate of neighbor joining is as follows.

THEOREM 1.1 *[From [1]] Let (T, M) be a General Markov Model tree with n leaves, with $0 < f \leq \lambda_e \leq g < \infty$ for all edges e in T. Let $\epsilon > 0$ be given, and let $\lambda^* = max_{ij}\{\lambda_{ij}\}$. Then there is a constant $C > 0$ such that, if the sequence length exceeds*

$$C \log n e^{O(\lambda^*)}$$

then, with probability at least $1 - \epsilon$, the Neighbor-Joining method recovers the true tree.

Note that $\lambda^* \leq g \cdot diam(T)$, where g is the maximum edge length and $diam(T)$ is the number of edges in the longest path in the tree T (i.e. it is the topological diameter of T), and that $diam(T)$ can be as large as $n - 1$. Thus the sequence length requirement of the Neighbor Joining method is bounded from above by a function that grows *exponentially* in n, even when g is fixed.

1.2.5 Fast-converging methods.

Since letting f be arbitrarily small or g be arbitrarily large affects the sequence length requirement, we are interested in developing methods for which polynomially long sequences ensure accuracy under the General Markov model, when both f and g are fixed, but arbitrary. In order to define this concept precisely, we first parameterize the General Markov model.

DEFINITION 1.3 $\mathrm{GM}_{f,g}$ contains those $(T, M) \in \mathrm{GM}$ for which $f \leq \lambda(e) \leq g$ holds for all edges $e \in E(T)$.

We now define absolute fast convergence:

DEFINITION 1.4 A phylogenetic reconstruction method Φ is *absolute fast-converging (afc)* for the GM model if, for all positive f, g, ε, there is a polynomial p such that, for all (T, M) in the GM model, on set S of n sequences of length at least $p(n)$ generated on T, we have $Pr[\Phi(S) = T] > 1 - \varepsilon$.

Note that method M operates without any knowledge of parameters f or g—or indeed any function of f and g. Thus, although the polynomial p depends upon both f and g, the method itself does not.

There are now several methods which have been proven to be afc (see [5, 6, 44, 31]), and in Section 1.7 we will describe how we derive one such afc method through the use of a DCM. All afc methods, whether implicitly or explicitly, have two phases: first they produce a set of trees, and then they select the best tree from the set. Proofs that the methods are afc then require proving (a) that the first phase produces a set that includes the true tree with high probability, given polynomial length sequences, and (b) the second phase picks the true tree with high probability, under the assumption that the true tree is in the set and the input sequences are of polynomial length.

1.3 The basic divide-and-conquer strategy

1.3.1 Introduction.

Each DCM is fundamentally based upon a divide-and-conquer strategy, which has the following three phrase structure:

- Phase I: Compute a decomposition of the dataset into overlapping subsets, and construct trees on the subsets using the base method.
- Phase II: Use a supertree method to merge the trees on the subsets into a tree on the full dataset
- Phase III: If the tree obtained in Phase II is not fully resolved (i.e. if the tree is not a binary tree), we resolve it further into a binary tree so that it optimizes the desired objective criterion (e.g., maximum parsimony).

We have designed these phases so that we can guarantee accuracy in the reconstructed tree obtained at the end of the first two phases under certain conditions. These guarantees and other properties of our DCMs rely upon the theory of triangulated graphs which is presented in Section 1.4 below. In general, however, we are motivated not only by theory but also by empirical performance, and so much of our discussion here will attempt to reflect those dual concerns.

1.3.2 Phase I: a brief overview.

The main issue in the design of Phase I is the decomposition of the set of taxa into overlapping subsets. Our approach for Phase I is to first construct a triangulated graph (that is, a graph without any simple induced cycle of size four or more) whose vertex set corresponds to the input set of taxa, and then compute a decomposition of the vertices of the triangulated graph (and hence of the set of taxa) into overlapping subsets. We describe how we obtain triangulated graphs from our input sets, and how we decompose these triangulated graphs, in Section 1.4 below.

1.3.3 Phase II: Merging the subtrees.

After Phase I is completed, we have a set of trees, one for each of the sets in the decomposition of the set of taxa. These subtrees share taxa in common, and the objective is to merge these subtrees into a tree on the full dataset. We would like this merger to retain accuracy if possible, so that in particular if all the subtrees are correct (meaning that they accurately reflect the true tree restricted to the subset), then the merger of these subtrees should be the true tree. This is the **Subtree compatibility problem:**

- **Input:** set $\mathcal{T} = \{T_1, T_2, \ldots, T_k\}$ with T_i an unrooted tree on leaf set S_i.
- **Output:** Tree T on leaf set $S = \cup_i S_i$, if it exists, such that $T|S_i = T_i$.

This problem is NP-hard, as was shown in [39]. Thus, any method for this problem which attempts to retain accuracy is likely to fail under some conditions, or to require exponential time.

The method we use is the *Strict Consensus Merger*, described originally in [20]. This is a polynomial time method which is based upon the theory of triangulated graphs, and has provable accuracy under certain conditions.

Strict Consensus Merger.

The Strict Consensus Merger of a set of trees is a technique we developed for use with subtrees obtained through our DCM decompositions. Details of this technique are given in [20, 21, 44]; here we provide a brief description.

The Strict Consensus Merger (SCM) operates by sequentially merging pairs of subtrees until all the subtrees have been merged into a tree on the full set, and the particular order in which the subtrees are merged matters. Given two trees t_1 and t_2 on $S_1 \subseteq S$ and $S_2 \subseteq S$, respectively, SCM operates as follows. First SCM computes $S_1 \cap S_2$, the set of leaves that the two trees share, and considers the two trees restricted to just that common set of leaves. The *strict consensus* of these two subtrees (i.e., the most resolved common contraction of the two trees) is then computed; this constitutes the "backbone" of the resultant tree. The remaining pieces of t_1 and t_2 (on leaf sets $S_1 - S_2$ and $S_2 - S_1$, respectively) are then reattached onto the backbone. If both t_1 and t_2 should contribute pieces to the same edge of the backbone, then that edge is bisected, and all the pieces of both trees are attached to that newly introduced node.

Note the following. First, the SCM of a set of trees can be computed in polynomial time, since the strict consensus of two trees is a linear time operation. Also, the SCM of a set of a set of trees is typically not a binary tree, since any conflict in the trees will result in edge contractions during the merger of the trees together. Finally, even if the set of trees is compatible (meaning that a supertree exists consistent with all the given trees), the SCM of these trees may not produce such a compatible supertree. This last comment is not surprising since the subtree compatibility problem is NP-hard [39], and hence a polynomial time algorithm cannot be expected to solve the problem. On the other hand, we showed in [20] that when the subtrees are "big enough" (a statement we quantify exactly) and the subtrees are compatible, then SCM *does* solve the subtree compatibility problem.

1.3.4 Phase III: Refining trees.

The strict consensus merger contracts edges in the subtrees in order to make the subtrees compatible with each other; as a consequence, the tree returned in Phase II is often not fully resolved. We therefore apply techniques for refining the tree obtained in Phase II. A typical approach for this refinement phase is to attempt to find a refinement of the given tree that optimizes some criterion, such as the maximum parsimony criterion, among all refinements. However, such problems tend to be NP-hard (see [4] for this problem when the optimization criterion is maximum parsimony). Heuristics for refining trees so as to optimize maximum parsimony are implemented in the major phylogeny software packages, but are not particularly effective nor fast. Consequently, the optimal tree refinement (OTR) problem is of general importance in phylogeny reconstruction.

1.4 Triangulated graphs

We now turn to the basic theory of triangulated graphs. The particular properties of triangulated graphs allow us to design these first two phases with provable performance guarantees in terms of accuracy of the reconstructed tree, and in terms of running time. (The interested reader is directed to Golumbic's excellent book [15] from which much of this theory can be obtained.)

1.4.1 Basic material.

We begin with a definition.

DEFINITION 1.5 A graph which has no induced simple cycles of length greater than three is a **triangulated graph**.

Triangulated graphs are perfect graphs and are well-studied (see [15] for more on triangulated graphs). The first basic theorem about triangulated graphs is that each triangulated has a perfect elimination scheme:

DEFINITION 1.6 A **perfect elimination scheme** for a graph $G = (V, E)$ is an ordering of the vertices $v_1, v_2, \ldots, v_n$, so that for each $i = 1, 2, \ldots, n-1$, $X_i = \Gamma(v_i) \cap \{v_{i+1}, v_{i+2}, \ldots, v_n\}$ is a clique (here $\Gamma(v_i)$ indicates the neighbor set of v_i).

Not only does every triangulated graph have a perfect elimination scheme, but such an ordering can be found in polynomial time. Using the existence of a perfect elimination scheme, the following two theorems (which are the basis of how we obtain the decomposition of S into overlapping subsets) can also be proved. The first result is as follows:

THEOREM 1.2 *Every triangulated graph $G = (V, E)$ has at most $n = |V|$ maximal cliques, and these can be found in $O(n^2)$ time.*

THEOREM 1.3 *For every triangulated graph $G = (V, E)$ $\exists X \subseteq V$ such that X is a clique and $G - X$ is the disjoint union of components $C_1, C_2, \ldots C_k$. Furthermore, we can find such an X such that X minimizes $max_i|C \cup X|$ in $O(n^3)$ time.*

A basic aspect of the design of a DCM is producing a triangulated graph. Here we describe two such ways of obtaining triangulated graphs.

1.4.2 Threshold graphs.

Let S be a set of taxa, and let $[d_{ij}]$ be a distance matrix for the set of taxa, and let q be any non-negative real number. Then the threshold graph for d and q is defined as follows:

DEFINITION 1.7 The threshold graph $TG(d, q)$ has vertex set S and edges (s_i, s_j) such that $d_{ij} \leq q$.

Therefore, if $q \geq \max d_{ij}$ then $TG(d, q)$ is a clique, and for small enough q (for example,

$q = 0$), the threshold graph will not be connected.

Before describing how we get triangulated graphs from threshold graphs, we need to define additive distances.

DEFINITION 1.8 An $n \times n$ matrix $[d_{ij}]$ for which there exists a tree T with n labelled leaves, with positive edge-weighting $w : E(T) \rightarrow R^+$ so that $d_{ij} = \sum_{e \in P_{ij}} w(e)$ for all i, j (where P_{ij} is the path in T between the leaves i and j) is said to be **additive**.

In [35] we proved the following:

THEOREM 1.4 *Let d be an additive matrix, and let q be a real number. Then $TG(d, q)$ is triangulated*

1.4.3 Short subtree graphs.

Let S be a set of taxa, and let T be a tree leaf-labelled by S, with non-negative edge-weights $w(e)$ assigned to each edge e in T.

DEFINITION 1.9 Let e be an edge of an edge-weighted binary tree T. Let t_1, t_2, t_3, and t_4 be the four subtrees around the edge e (i.e., t_1 through t_4 are the components of $G - \{x, y\}$, where $e = (x, y)$). Let x_i denote those leaves in t_i which are closest to the edge e (using the path lengths defined by the edge-weighting on T). Then the **short subtree around** e is $x_1 \cup x_2 \cup x_3 \cup x_4$.

We now define the short subtree graph.

DEFINITION 1.10 Let T be a tree with leaf set S and edge weighting $w : E(T) \rightarrow R^+$. Let G be the graph with vertex set S and edge set E defined by $(s_i, s_j) \in E$ if and only if $\exists e \in E(T)$ such that s_i and s_j are both in the short subtree around e. This is the **short subtree graph** of (T, w), denoted by $SSG(T, w)$.

In [35] we showed the following:

THEOREM 1.5 *Let T be any tree with positive edge-weighting w. Then the short subtree graph defined by T and w is triangulated.*

1.4.4 Decompositions of triangulated graphs.

Theorems 1.2 and 1.3 imply two decompositions of the vertex set of a triangulated graph G, as follows:

- **Max-clique decomposition**: given a triangulated graph G, return the set of maximal cliques of G.
- **Separator-Component decomposition**: Given G, find a clique separator X and compute all the components of $G - X$. Then return the sets of the form $X \cup C$, where C is one of the components of $G - X$.

The first type of decomposition is uniquely determined by the triangulated graph, and for this reason we will refer to it as "*the* max-clique decomposition". However, a triangulated graph can have many different clique separators, each of which thus can define a different decomposition. For this reason, we will refer to any decomposition obtained by choosing a clique separator as "*a* separator-component decomposition". In Theorem 1.3 we showed that picking a clique separator X so as to minimize the maximum size of any created subproblem could be solved in $O(n^3)$ time, but this running time is not always acceptable. Therefore, we may sometimes prefer to use a suboptimal separator-component decomposition, i.e. one that would produce somewhat larger subproblems, if it can be computed faster.

Now consider the difference between the max-clique decomposition and a separator-component decomposition on the same fixed triangulated graph G. It is easy to see that the max-clique decomposition will produce more subproblems (up to n of them, where $n = |V|$), but each subproblem will be smaller (or at least not larger) than the subproblems obtained by the separator-component decomposition. However, the separator-component decomposition will produce potentially only a few subproblems. Furthermore, the pairwise intersections of the subsets produced by the max-clique decomposition can differ significantly (and will even be disjoint in some cases), whereas in any separator-component decomposition all pairwise intersections are the same. These differences will be significant in developing DCMs for different base methods.

1.5 Designing DCMs

1.5.1 Introduction.

All of our DCMs use the same three phase structure (although some also use recursion and/or iteration), with the main difference between the DCMs being the decomposition technique. All current DCMs first construct a triangulated graph and then apply either the max-clique or a separator-component decomposition to the graph to obtain subproblems. The combination of base method, choice of triangulated graph, and decomposition technique on that triangulated graph, impact the behavior of the resultant "DCM-boosted" method. For example, methods which will take a long time on big datasets will finish faster on the max-clique decomposition. A more subtle point is the impact of error on subsets – since the technique we use in merging subtrees contracts edges whenever two subtrees disagree, there is a potential for a greater loss of resolution in the max-clique decomposition than in a separator-component decomposition, especially when the separator is small. The difference between using threshold graphs and short subtree graphs is also interesting, but depends as much on the dataset as on the method. Thus, the design of a DCM reflects the particular properties of the base method and of the particular dataset, and only by studying the actual performance of the resultant DCM-boosted methods can we tell which design strategy will be the most beneficial.

We begin this section with a description of how we obtain triangulated graphs from molecular datasets.

1.5.2 Obtaining triangulated graphs from datasets.

Threshold graph decompositions.

Threshold graph decompositions can be used on any dataset for which a distance can be defined between each pair of taxa. In molecular sequence datasets, these distances can be Hamming distances, or distances obtained under some statistical estimation procedure selected to match the model of evolution underlying the dataset.

In a threshold graph decomposition, we are given a set S of sequences and a matrix $[d_{ij}]$ of distances on the set S of taxa. To compute a threshold graph we must first select the threshold (the value q). We then construct the threshold graph, $TG(d,q)$ (see Definition 1.7). If $TG(d,q)$ is triangulated, we can compute either the max-clique decomposition, or a separator-component decomposition; however, if $TG(d,q)$ is not triangulated, then we must first triangulate it, by adding edges to $TG(d,q)$ so that the graph is triangulated. However, when we add edges to $TG(d,q)$ we affect the decompositions (either max-clique or separator-component) that we can obtain from the triangulated graph.

In some of our DCMs our objective is to minimize the maximum evolutionary distance within any subproblem, so that in a max-clique decomposition we would wish the cliques to have the smallest maximum distance. This suggests the following objective in the triangulation process: add edges to $TG(d,q)$ so as to minimize the "weight" of the heaviest edge added. This optimization problem is NP-hard, however, and so in our experiments, we have used a greedy triangulation scheme that works reasonably well: compute for each vertex v in the graph the value $W(v) = max\{d_{ij} : \{i,j\} \subseteq \Gamma(v)\}$, where $\Gamma(v)$ denotes the neighbors of the vertex v. Select the vertex v that minimizes $W(v)$ and make it simplicial (i.e., make the neighbors of v into a clique). Recurse on $G - \{v\}$. This approach produces a triangulation of G but may not minimally triangulate the graph G; however, in our experiments this worked quite well. However, see also [25] for a polynomial time technique that creates an acceptable triangulation.

Thus, threshold graph decompositions have the following structure. They begin with a distance matrix $[d_{ij}]$, and operate as follows:

1. Pick a threshold $q \in \{d_{ij}\}$
2. Construct $TG(d,q)$
3. Add edges to $TG(d,q)$ to triangulate it, producing graph G
4. Compute either the max-clique or a separator-component decomposition from G

Guide tree decompositions:

We now describe how we can obtain guide trees, and from them the triangulated short subtree graph.

The most typical technique for obtaining a guide tree is to use some phylogeny reconstruction method (such as a heuristic for maximum parsimony or maximum likelihood, or perhaps a fast method such as neighbor joining) to obtain an estimation T of the true tree. Given T, we can then use one of many techniques to assign edge lengths. For example, if the set of taxa are biomolecular sequences in a multiple alignment, then we can assign edge lengths to T by using the Fitch-Hartigan dynamic programming fixed-tree maximum parsimony algorithm of [12] to assign sequences to internal nodes, and then use Hamming distances to define edge lengths. We can also use maximum likelihood estimation of edge lengths, which may be more accurate but will take more time than maximum parsimony. In general, however, if T was obtained using a phylogeny reconstruction method, it will typically already have edge lengths (such is the case with the three techniques we mentioned earlier - heuristic MP, heuristic ML, or neighbor joining).

Given the guide tree and T with its edge lengths, we then compute the short subtree graph. This is easily done in polynomial time. Once the short subtree graph is obtained, we can compute either the max-clique or a separator-component decompositions on it (since it is already triangulated). As noted before, finding an optimal separator-decomposition - although polynomial time - can be more expensive than desired; consequently faster decompositions based upon clique-separators can also be used. In [34] we showed how we could compute a decomposition we call the "heuristic centroid-edge" decomposition in linear time (this fast running time was accomplished without explicitly constructing the short subtree graph). This decomposition worked very well in practice, as we showed in [34, 35].

1.5.3 Considerations in design strategies.

We have described ways we can obtain triangulated graphs, and ways we can decompose a triangulated graph. How do these choices interact with base methods?

Some methods - in particular, distance-based methods like neighbor joining [37], have poor topological accuracy on datasets with large evolutionary diameters, although they are quite fast. These methods would therefore seem to benefit from decompositions that produce the smallest diameter subproblems. Other methods, in particular exhaustive searches for optimal trees under hard optimization criteria, can only realistically handle quite small datasets – maximum parsimony is limited to perhaps 20 or 25 taxa, and maximum likelihood limited to much smaller datasets; these methods would require subproblems to be as small as possible. In a third class are local-search heuristics (like the hill-climbing heuristics used in maximum parsimony searches), which seem not to be impacted by large diameters so much, but are still impacted by dataset size. Understanding the best design strategy for these local-search heuristics is more complicated.

The particular technique used to obtain a triangulated graph also has an impact on the resultant DCM. If we use a guide tree, there is only one triangulated graph that we can obtain, but different guide trees will produce potentially different decompositions. This makes guide tree decompositions useful for heuristic searches for optimal trees under criteria such as maximum parsimony, because as the search finds better solutions, it can become a new guide tree, and a new decomposition can be obtained.

The issues involved in selecting threshold graph decompositions are more complicated. In this case, the distance matrix $[d_{ij}]$ is usually considered fixed, but the threshold can change. If the threshold is too small, the threshold graph will not even be connected, and so the tree on the full dataset cannot be reconstructed from subtrees, even if they are correctly computed. If the threshold is too big, the subproblems become essentially as difficult (almost as large and with almost the same evolutionary diameter) as the full dataset, although correct subtrees on the subproblems would then be likely to define the full tree. Thus, finding the "correct" threshold to use is a difficult problem.

Our experiments with real and simulated data showed us two very interesting things. The first was that the threshold graphs obtained for biomolecular sequence datasets had very large cliques – so large, in fact, that on many datasets the largest subproblems obtained in a threshold graph decomposition were close to the full dataset size [35, 34]. This meant that we'd be analyzing several subproblems of size almost the full dataset size, with the consequence that there was little gained in using a threshold graph decomposition. On the other hand, the subset sizes obtained by using reasonable guide trees (obtained using good heuristic searches for MP, for example) produced much smaller subproblems for the same datasets [35, 34], so that decompositions based upon the short subtree graph were more suited for boosting heuristic searches for maximum parsimony or maximum likelihood.

Other experiments showed that even the best distance-based methods (neighbor joining, for example) had poor topological accuracy on large diameter subproblems, confirming the theory that had been established for the convergence rates (i.e., the sequence length requirements) of these methods [1]. In order to develop methods with provably good convergence rates, we needed to work with threshold graphs, rather than with guide trees. In order to obtain the provable theory, however, we had to take a third approach - rather than selecting a threshold, we compute all possible threshold graphs (one for each threshold that creates a connected graph), and hence we compute $O(n^2)$ trees, one for each threshold graph. We then have to apply yet another step (whereby we obtain a single tree from the $O(n^2)$ trees) in order to return a tree for the input taxa.

Finally, DCMs have also been designed for reconstructing phylogenies on whole genomes (using gene order data) using the GRAPPA software suite (see [29] for a description of GRAPPA and of the DCM-boosting designed for use with GRAPPA techniques). This is empirically a harder problem – datasets above 13 or so genomes cannot be handled by GRAPPA, and so decomposing the dataset into smaller subsets (each subset explicitly limited to at most 13 genomes) is an absolute requirement.

In the next sections we describe the DCMs we used for boosting maximum parsimony heuristics and for use with distance-based methods; readers interested in the use of DCMs for boosting GRAPPA should see [29].

1.6 DCM-boosting techniques for maximum parsimony

1.6.1 Objectives.

We begin with a description of our DCMs for maximum parsimony.

The main issue confronting maximum parsimony and maximum likelihood heuristics is running time – they take a long time to reach reasonable accuracy on large datasets. In fact, it is not uncommon for phylogenetic analyses to take weeks or months (or years) on large datasets – and with the increasing availability of sequence data, this trend may continue or even get worse. Finding ways to make these analyses much faster is the objective of this algorithm design.

The simplest design of a DCM involves only one application of a base method on a subproblem - so that after the subtrees are constructed and merged together, the analysis stops. When the objective is to solve an optimization problem, however, this no longer makes sense - we will want to continue searching for better trees as long as the data suggests that we may not have found sufficiently good trees. Therefore, we would usually use the output of a DCM as the input to a standard heuristic search, with the hope that the improvement obtained by the DCM would give us a "head start" over standard approaches. In addition, we would like to design DCMs that allow for iterative use, so that instead of switching over to standard heuristic searches we could continue using DCM-boosted heuristics. Thus, iterative-DCMs will be potentially beneficial for maximum parsimony (as well as for other optimization problems).

Thus, DCMs which produce smaller subproblems make a lot of sense, and it would seem that the smallest possible subproblems would be the most desirable. However, in our experiments, the loss of resolution that results from using the max-clique decomposition proved to be more of a problem than the running time used in a separator-component decomposition, because the third phase in which we attempt to resolve the tree optimally with respect to maximum parsimony was too expensive. Furthermore, we observed that the subsets obtained using the threshold graph decomposition produced subproblems that were quite big - 90% of the taxa in the largest subproblem - so that there was little improvement gained in the running time.

For this reason, we developed the short subtree graph decomposition, since these produced maximal cliques that were much smaller than the subproblems we obtained in the threshold graph decompositions. What seems to have worked quite well for maximum parsimony heuristics is to produce a separator-component decomposition on a short subtree graph: these subproblems contain no more than 50-60% of the taxa, and so are substantially smaller than the subproblems obtained using the threshold graph decomposition. The separators we find in these decompositions tend also to be very small – four or five taxa, in fact. Furthermore, because these subproblems overlap only on the separator, very little resolution is lost during the merger of the subtrees into the supertree. Finally, when 50-60% of the taxa is still too large for the base method, we examined recursive uses of the decomposition.

1.6.2 DCM3.

Thus, we have several variants of a basic design, which we call DCM3. In its simplest form, we would use some technique to obtain a guide tree, and then compute the short subtree graph from the guide tree. We would then compute some separator-component decomposition based upon the guide tree, thus producing a set of subsets of the original taxa. Following this, we would apply the base method to the subproblems, merge the resultant subtrees using the strict consensus merger, and then refine the tree. In summary, the basic algorithm is as follows.

$DCM3(T_1, w)$.

The input to this routine is a tree, T_1, leaf-labelled by a set S of sequence, and an edge weighting w of T_1.

1. Construct $G = SSG(T_1, w)$, the short subtree graph based upon the guide tree T_1 with is edge-weighting w. Compute a separator-component decomposition on G, producing subsets $S_i = C_i \cup X$, where X is the separator and C_i is the i^{th} component of $G - X$.
2. Let t_i be $T_1|S_i$ (that is, t_i is the subtree of T_1 induced by the set S_i of taxa). Use a preferred method to construct trees on each subset, starting with the tree t_i for the subset S_i.
3. Merge the resultant subtrees using the Strict Consensus Merger.
4. Return a refinement of the resultant tree.

Several comments are worth making here. The first is to note that we do not simply apply the favored heuristic to each subproblem, but rather we take advantage of our current best tree (the guide tree, that is) in order to initialize the search at a (hopefully) good tree. That is, we will begin our search for the optimal tree on the subset S_i with the tree t_i.

Secondly, we have left open several steps of the technique: for example, which separator-component decomposition do we compute?, how do we refine the resultant tree?, how exactly do we obtain the guide tree T_1?, and how long do we apply our favored heuristic on each subset S_i? These are all aspects of the design of DCM3 that will depend quite strongly upon the particular base method, as well as the properties of the dataset being analyzed.

Furthermore, we can choose different MP heuristics for each of the steps (one for the initial step where we obtain our tree T_1, and another where we analyze subsets). Also, we can try to optimize the decomposition as described above or just take some reasonably good decomposition, and we can attempt to optimally refine the unresolved tree we get from the Strict Consensus Merger, or we can refine it heuristically (or even randomly). Thus, the particular application of this DCM3 strategy involves decisions at various points.

1.6.3 Iterative-DCM3.

The main use we have made of DCM3 is in its iterative form, where we have alternated between the use of a heuristic on the full dataset and the use of the same heuristic (perhaps applied slightly differently) on the subproblems. That is, Iterative-DCM3 follows upon the construction of an initial guide tree (T_1), and has the following steps.

- Repeat until you want to stop:
  - Let $T = DCM3(T_1)$ (i.e. T is the tree obtained by applying DCM3 to the guide tree T_1)
  - Apply your favored heuristic to T until you reach a local optimum, or until you satisfy some stopping rule, and let T_1 be the current best tree.

Note that this use of DCM3 introduces yet another level of flexibility (or ambiguity): now we have three places where we will apply a heuristic for MP to a dataset: the initial stage, where we obtain our first estimate of the optimal tree from scratch, and then we will alternate between applying a heuristic just to subsets, and applying a heuristic to the full set. We may elect to use the same basic heuristic, but apply it with fewer or more iterations depending upon the size of the subset being analyzed. Or, we may change the heuristic we use (and not just vary it by changing the number of iterations) depending upon the size or features of the subset. These issues again will depend upon the features of the dataset and the optimization criterion, as well as upon the desired level of accuracy and/or the amount of time that is available for the analysis.

1.6.4 Recursive DCM3.

Recursive DCM3 is a simple modification of DCM3, in which we recursively decompose subsets until we reach a desired subset size. Then we apply the favored heuristic to the subsets (starting, as before, with the subtree induced by the guide tree on each subset), and then merge trees using the strict consensus merger as we go back up the recursion tree. Once all the subtrees are merged into a tree on the full dataset, we then apply the refinement step.

The main advantage of Recursive-DCM3 is that the subproblems can be made significantly smaller, even with just one or two levels of recursion.

1.6.5 Recursive-Iterative-DCM3.

Recursive-Iterative-DCM3 combines both recursion and iteration, so that we iteratively call Recursive-DCM3. This technique has the best performance of the various techniques we examined in our studies, when the base methods were standard heuristics for maximum parsimony.

1.6.6 Experimental results.

In this section we summarize the results of fairly extensive studies on real datasets ranging from 1000 sequences up to almost 14,000 sequences, with base methods for maximum parsimony taken from different software packages. An example of one such study is given in Figure 1.6.6.

We have experimented with several variants of DCMs for use with heuristics for maximum parsimony on a number of very large datasets. These experiments have examined base methods and compared them to their DCM-boosted versions on a number of real datasets. We included DCM2, which is the separator-component decomposition based upon the threshold graph obtained using the smallest threshold that produces a connected graph [21]. In order to compare DCM2 and DCM3 to base heuristics, we used the output of DCM2 and DCM3 as a starting tree for their base heuristics, so that we could explore performance over a longer period of time. Thus, the comparison is made as a function of time – examining the best MP score found at each point in time, over a period of days or weeks (depending upon the dataset). We consider DCM-boosting to be advantageous if we obtain an improvement at every point in time for a considerable length of time, preferably for at least 24 hours. These experiments (detailed in [34, 35]) showed the following:

- The better the base method is, the better the DCM must be in order to obtain an advantage over the base method. Thus, even a poor DCM can improve upon a poor base method, but for the best base methods, we need very good DCMs to obtain an advantage.
- DCM2 (the separator-component decomposition applied to a threshold graph) produces subproblems that are very large, and the decomposition takes a long time. In fact, when we use very good heuristics for MP, DCM2-boosting worsens the performance rather than improving it. Consequently, DCM2 is not helpful for solving maximum parsimony on most datasets we examined, by comparison to good base heuristics.
- DCM3, based upon finding an optimal decomposition but using a random refinement, and then continuing with the base heuristic, did not typically improve the performance of the best base methods, but it also didn't generally make things worse.
- Recursive-DCM3 gave a slight advantage over the best base heuristics, and a somewhat larger advantage over other base methods.
- Iterative-DCM3 gave a somewhat larger advantage over all base methods than Recursive-DCM3.
- Recursive-Iterative-DCM3, using both optimal and heuristic decompositions, gave the largest advantages over even the best base heuristics.
- All advantages obtained by any DCM-boosting technique depended on the difficulty of the dataset and the quality of the base heuristic. Thus, when used with good base heuristics on small to moderately large datasets, the advantage is not always significant. Thus, the main advantage obtained is on the largest and most difficult datasets.

1.7 DCM-boosting distance-based methods

1.7.1 Objectives.

In this section we describe the DCM we have developed for use with distance-based reconstruction methods.

The best distance-based methods are very fast, and topologically very accurate even on large datasets as long as the subproblems have small diameter. However, as the diameter increases, their accuracy decreases (see [31, 28, 32, 30]. This empirical observation is supported by the theory that has been established about these methods as described earlier in the chapter.

Thus, the main empirical purpose in devising a DCM for use with neighbor joining is to produce a method that will enable recover the true tree from shorter sequences (as well

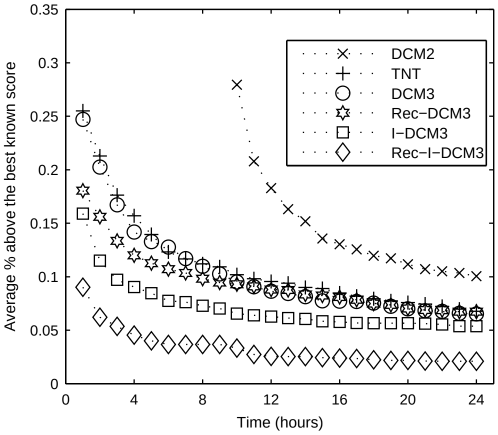

FIGURE 1.1: The performance of different versions of DCM3(TNT) on a dataset of 4114 aligned 16s rRNA Actinobacteria sequences, each of length 1263 (from the Ribosomal Database Project at the University of Michigan). The x-axis reports the best MP score found by each method during the first 24 hours of analysis. The y-axis measures the error rate of each method as a percentage above the current best MP score found on this dataset, where "current best" means the best score found using any method over any amount of time. The base method is the Ratchet heuristic for MP within the TNT software [14]. The recursive-DCM3 code produces subsets of size 1/8th the original dataset size. Each point is the average of 5 runs, and all runs are done on the same machines – 700MHz Pentiums.

as more accurate trees at every sequence length). However, the main theoretical purpose is to use DCM-boosting so as to produce an absolute fast converging (afc) method from NJ (see Definition 1.3). The design of the DCMs for use with NJ (and other distance-based methods) differ slightly depending upon whether the empirical goal or the theoretical objective is more important.

1.7.2 $DCM1_{NJ} + SQS$: Designing an afc method using DCM-boosting.

In [44] we gave a technique to produce an absolute fast converging method from any method with an exponential convergence rate, such as neighbor joining. Recalling the definition of the threshold graph given in Definition 1.7, the DCM1 decomposition, and the Strict Consensus Merger, the technique is as follows:

- Phase I:
  - For each $q \in \{d_{ij}\}$, compute $t_q = DCM1(TG(d, q))$; i.e.,
    - Construct the threshold graph $TG(d, q)$, and triangulate it with a minimum weight triangulation (minimizing the maximum weight of any added edge).
    - Compute the max-clique decomposition of $TG(d, q)$, and construct trees on each maximal clique using neighbor joining.
    - Merge the trees together using the Strict Consensus Merger.
- **Phase II:**

- Evaluate each tree t_q according to the Short Quartet Support Criterion (see [44]).
- Return the tree with the best score.

We have explicitly referenced the neighbor-joining (NJ) method in this description, but any method could be used in its place as a base method. However, in order to obtain a theoretical guarantee of fast convergence, the base method needs to have a convergence rate that is no worse than exponential in the maximum evolutionary diameter of the model tree (this is true for NJ and most of the other distance-based methods that have been proposed).

Note that this DCM requires that the triangulation of the threshold graph be optimal, which requires therefore solving an NP-hard problem. However, in [25] Jens Lagergren showed that certain polynomial time triangulation techniques also have the desired theoretical properties so that his variant of DCM-boosting would also produce an afc method without requiring an exact solution to the NP-hard problem of minimally triangulating the threshold graph.

1.7.3 Improving the empirical performance of $DCM1_{NJ} + SQS$.

$DCM1_{NJ} + SQS$ is designed for theoretical performance, and thus involves more computational time than we would want (in particular, it involves solving a hard computational problem and it produces $O(n^2)$ trees in Phase I). Improving upon the speed of this DCM is a necessary objective if this technique is going to be useful for any real analysis. Furthermore, the specific technique used in Phase II (selecting the tree which optimizes the Short Quartet Support) was also used because of its theoretical properties, but that technique - although theoretically optimal - turns out not to have as good performance (as shown in our simulation studies) as other criteria. Thus, in a later study [31] we explored the empirical consequences of modifying the algorithmic design of $DCM1_{NJ} + SQS$, in order to improve the speed and/or accuracy in simulated and real datasets. We specifically examined a variant where we modified Phase I by examining only ten thresholds (rather than all possible thresholds), and where we used a greedy technique to triangulate the threshold graph (as described earlier in this chapter). We then modified Phase II by selecting the tree that optimized some other criterion (we examined specifically maximum parsimony or maximum likelihood, and found them largely to have the same performance and both superior to SQS). These studies led us to propose (as heuristics) two DCMs for boosting NJ: $DCM1_{NJ} + MP$ and $DCM1_{NJ} + ML$, neither of which has provable theoretical performance, but both of which provide a distinct topological accuracy advantage over NJ, while being still reasonably fast. (Neither is as fast as NJ, but both are fast enough to complete analyses in a few minutes rather than in hours or days, as any serious MP or ML analysis would require.)

1.7.4 Experimental Results.

We designed a simulation study to explore the relative performance of our heuristic approximations to $DCM1_{NJ} + MP$ and $DCM1_{NJ} + SQS$, in comparison to NJ and to a provably afc method called HGT+FP (for "Harmonic Greedy Triplets, plus the Four Point Method) [8, 7, 6], a sample of which is shown in Figure 1.7.4. (See [28, 32, 30, 36] for some more experiments.) Our two methods are thus not provably afc but rather only heuristic approximations to $DCM1_{NJ} + SQS$, which is afc.

Our implementation of $DCM1_{NJ} + SQS$ is heuristic because we only examine ten thresholds rather than all possible thresholds, and we do not optimally triangulate the threshold graphs, thus it is not provably afc. Even perhaps more serious, from a theoretical viewpoint, the use of MP as the selection criterion in the second phase of $DCM1_{NJ} + MP$ automatically makes the method not afc and probably not even provably statistically consistent.

However, the difference in performance between the methods is striking. The error rates of NJ rises quite quickly, but the remaining methods have flat error rates within these dataset sizes. Furthermore, using MP rather than SQS *improves* the performance of $DCM1_{NJ}$, even though from a theoretical perspective it is worse. Finally, although HGT+FP is provably afc its performance is worse than the $DCM1_{NJ}$ methods at all dataset sizes.

The difference in performance between these methods is due to a combination of factors. NJ's performance problem is due to the fact that it uses all the entries of the matrix, without sufficiently downweighting large entries, and hence has an exponential convergence rate. The difference in performance between HGT+FP and our $DCM1_{NJ}$ methods is largely due to the improvement obtained in practice by using NJ rather than a quartet-based method, which is what HGT+FP essentially uses. The difference in performance between $DCM1_{NJ} + SQS$ and $DCM1_{NJ} + MP$ is more mysterious: why should MP, which is not statistically consistent, outperform SQS as a selection criterion? Once again, the difference may be in the specific design of SQS; it is based upon quartet accuracy, with a particular technique to determine the "correct" tree on each quartet. Although this criterion is theoretically sound, empirically quartet-based methods (such as SQS) are not as accurate as methods that compute trees from all the data. Despite the theoretical guarantees that can be established for quartet methods, they have in general not been able to be as accurate in simulation as the neighbor joining method, as shown in [22].

The lesson from this comparison of distance-based methods is an interesting one, and instructive: while dividing into subproblems can yield improvements in accuracy, precisely how one divides into subproblems is tremendously important.

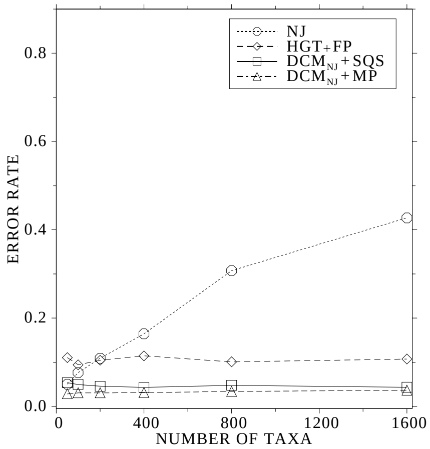

FIGURE 1.2: The performance of $DCM1_{NJ}$ with different techniques used in Phase II, compared to NJ and to another afc method, HGT+FP, as a function of the number of taxa. In this experiment we simulated evolution of sequences with 1000 sites down uniform distribution random trees with branch lengths drawn from the same distribution under the Kimura 2-parameter model [24] of evolution. K2P distances were used as inputs to each method.

1.8 Related work and conclusions

DCM-boosting has also been used to improve the speed of software for the inversion and breakpoint phylogeny problems (both NP-hard problems) within the GRAPPA software suite for whole genome phylogeny reconstruction [29]. There are two basic challenges for GRAPPA's phylogenetic analysis: first, because the techniques employed within GRAPPA exhaustively search all trees on each input dataset, GRAPPA is explicitly limited to analyzing datasets below (about) 14 taxa. Secondly, because of its design, GRAPPA is unable to efficiently analyze any dataset where the underlying tree has very large edge lengths. The authors used a DCM in order to address the first of these two challenges; the second remains a problem for any explicit attempt to solve these optimization problems.

The DCM they designed for use with GRAPPA [29] employed many levels of recursion, so that each subproblem was small enough. The specific design of their DCM was similar to the $DCM1$ design, but computed a consensus tree of the different trees (one for each threshold) they obtained, and they studied their DCM-boosting technique in simulation. Their study showed that using DCM-boosting allowed GRAPPA to be applied to datasets with thousands of taxa, in other words a huge improvement in performance.

Future research will explore DCMs for other types of base methods, such as maximum likelihood and phylogenetic sequence alignment [38].

Finally, we note that other divide-and-conquer strategies have been proposed. Some of the most well known are quartet-based methods, such as quartet puzzling [41], quartet-cleaning [3], the Q^* method [2], and the short quartet methods [9, 10], but none of these has been shown to reliably outperform neighbor joining (see [22] for some of these results). On the other hand, methods such as Compartmentalization [27] are also promising, and could be investigated. Research into supertree methods which can construct trees from arbitrarily defined subtrees also needs further investigation.

1.9 Acknowledgments

This work was supported in part by the National Science Foundation, the David and Lucile Packard Foundation, the Institute for Cellular and Molecular Biology at the University of Texas at Austin, the Radcliffe Institute for Advanced Study, and the Program for Evolutionary Dynamics at Harvard University.

References

References

[1] K. Atteson. The performance of the neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, 25:251–278, 1999.

[2] V. Berry and O. Gascuel. Inferring evolutionary trees with strong combinatorial evidence. In *Proc. 3rd Ann. Int'l Conf. Computing and Combinatorics COCOON97*, pages 111–123. Springer Verlag, 1997. in *LNCS* **1276**.

[3] V. Berry, T. Jiang, P. Kearney, M. Li, and T. Wareham. Quartet cleaning: improved algorithms and simulations. In *Proc. Europ. Symp. Algs. ESA99*, pages 313–324. Springer Verlag, 1999. in *LNCS* **1643**.

[4] M. Bonet, M. Steel, T. Warnow, and S. Yooseph. Faster algorithms for solving parsimony and compatibility. *J. Comput. Biol.*, 5:409–422, 1999.

[5] M. Cryan, L. Goldberg, and P. Goldberg. Evolutionary trees can be learned in polynomial time in the two-state general markov model. In *Proc. IEEE Symp. Foundations of Comput. Sci. FOCS98*, pages 436–445, 1998.

[6] M. Csűrös. Fast recovery of evolutionary trees with thousands of nodes. RECOMB, 2001.

[7] M. Csűrös and M. Y. Kao. $O(n^2 logL)$ - time accurate recovery of evolutionary trees with more than one thousand leaves: an experimental combination of harmonic greedy triplets and the minimum evolution principle, 1999. Preprint, Yale U.

[8] M. Csűrös and M. Y. Kao. Recovering evolutionary trees through harmonic greedy triplets. *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA99)*, pages 261–270, 1999.

[9] P.L. Erdos, Mike Steel, L. Székély, and T. Warnow. A few logs suffice to build almost all trees –I. *Random Structures and Algorithms*, 14:153–184, 1997.

[10] P.L. Erdos, Mike Steel, L. Székély, and T. Warnow. A few logs suffice to build almost all trees –II. *Theor. Comp. Sci.*, 221:77–118, 1999.

[11] J. Felsenstein. Cases in which parsimony and compatibility methods will be positively misleading. *Syst. Zool.*, 27:401–410, 1978.

[12] W. M. Fitch. Toward defining the course of evolution: minimum change for a specified tree topology. *Syst. Zool.*, 20:406–416, 1971.

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

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

[15] M. Golumbic. *Algorithmic graph theory and perfect graphs.* Academic Press Inc, 1980.

[16] D. M. Hillis. Inferring complex phylogenies. *Nature*, 383:130–131, 1996.

[17] D. M. Hillis, C. Moritz, and B. Mable. *Molecular Systematics.* Sinauer Pub., Boston, 1996.

[18] J. Huelsenbeck. Performance of phylogenetic methods in simulation. *Syst. Biol.*, 44:17–48, 1995.

[19] J. Huelsenbeck and D. Hillis. Success of phylogenetic methods in the four-taxon case. *Syst. Biol.*, 42:247–264, 1993.

[20] D. Huson, S. Nettles, and T. Warnow. Disk-covering, a fast-converging method for phylogenetic tree reconstruction. *Comput. Biol.*, 6:369–386, 1999.

[21] D. Huson, L. Vawter, and T. Warnow. Solving large scale phylogenetic problems using DCM2. In *ISMB99*, pages 118–129, 1999.

[22] K. St. John, B. M. Moret, L. Vawter, and T. Warnow. Large performance study of phylogenetic methods: (unweighted) quartet methods and neighbor-joining. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA 01)*, pages 186–195, 2001.

[23] J. Kim and T. Warnow. Tutorial on phylogenetic tree estimation. In *Proc. 7th Int'l Conf. on Intelligent Systems for Mol. Biol. ISMB99*, 1999.

[24] M. Kimura. A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *J. Mol. Evol.*, 16:111–120, 1980.

[25] J. Lagergren. Combining polynomial running time and fast convergence for the disk-covering method. *Journal of Computer and System Science*, 65(3):481–493, 2002.

[26] W. H. Li. *Molecular Evolution.* Sinauer, Massachuesetts, 1997.

[27] B. D. Mishler. Cladistic analysis of molecular and morphological data. *American Journal of Physical Anthropology*, 94:143–156, 1994.

[28] B.M.E. Moret, U. Roshan, and T. Warnow. Sequence length requirements for phy-

logenetic 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, 2002.

[29] B.M.E. Moret, J. Tang, and T. Warnow. Reconstructing phylogenies from gene-content and gene-order data. *Mathematics of Evolution and Phylogeny*, pages 321–352, 2005. (to appear).

[30] L. Nakhleh, B.M.E. Moret, U. Roshan, K. St. John, and T. Warnow. The accuracy of fast phylogenetic methods for large datasets. In *Proc. 7th Pacific Symp. Biocomputing (PSB'2002)*, pages 211–222. World Scientific Pub., 2002.

[31] L. Nakhleh, U. Roshan, K. St. John, J. Sun, and T. Warnow. 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 U. Press, 2001.

[32] L. Nakhleh, U. Roshan, K. St. John, J. Sun, and T. Warnow. The performance of phylogenetic methods on trees of bounded diameter. In *Proc. 1st Int'l Workshop Algorithms in Bioinformatics (WABI'01)*, volume 2149 of *Lecture Notes in Computer Science*, pages 214–226. Springer-Verlag, 2001.

[33] D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.

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

[35] U. Roshan, B. M. E. Moret, T. Warnow, and T. L. Williams. Rec-I-DCM3: a fast algorithmic technique for reconstructing large phylogenetic trees. In *Proceedings of the IEEE Computational Systems Bioinformatics conference (CSB)*, Stanford, California, USA, 2004.

[36] U. Roshan, B. M. E. Moret, T. L. Williams, and T. Warnow. Performance of supertree methods on various dataset decompositions. In O. R. P. Bininda-Emonds, editor, *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, volume 3 of *Computational Biology*, pages 301–328. Kluwer Academics, 2004. (Dress, A. series ed.).

[37] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, 4:406–425, 1987.

[38] D.D. Sankoff and R.J. Cedergren. Simultaneous comparison of three or more sequences related by a tree. In D. Sankoff and J. B. Kruskal, editors, *Time Warps, String Edits, and Macromolecules: the Theory and Practice of Sequence Comparison*, pages 253–264. Addison-Wesley, Reading, MA, 2003.

[39] M. A. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992.

[40] M. A. Steel. Recovering a tree from the leaf colourations it generates under a Markov model. *Appl. Math. Lett.*, 7:19–24, 1994.

[41] K. Strimmer and A. von Haeseler. Quartet puzzling: A quartet maximum likelihood method for reconstructing tree topologies. *Molecular Biology and Evolution*, 13(7):964–969, 1996.

[42] D. L. Swofford. PAUP*: Phylogenetic analysis using parsimony (and other methods), 1996. Sinauer Associates, Underland, Massachusetts, Version 4.0.

[43] C. Tuffley and M.A. Steel. Modelling the covarion hypothesis of nucleotide substitution. *Mathematical Biosciences*, 147:63–91, 1997.

[44] T. Warnow, B. M. Moret, and K. St. John. Absolute convergence: true trees from short sequences. *Proceedings of ACM-SIAM Symposium on Discrete Algorithms (SODA 01)*, pages 186–195, 2001.

Index