

# Performance of Supertree Methods on Various Dataset Decompositions

Usman Roshan\*    Bernard M.E. Moret<sup>†</sup>    Tiffani L. Williams<sup>†</sup>  
Tandy Warnow\*

## Abstract

Many large-scale phylogenetic reconstruction methods attempt to solve hard optimization problems (such as Maximum Parsimony (MP) and Maximum Likelihood (ML)), but they are severely limited by the number of taxa that they can handle in a reasonable time frame. A standard heuristic approach to this problem is the divide-and-conquer strategy: decompose the dataset into smaller subsets, solve the subsets (i.e., use MP or ML on each subset to obtain trees), then combine the solutions to the subsets into a solution to the original dataset. This last step, combining given trees into a single tree, is known as supertree construction in computational phylogenetics. The traditional application of supertree methods is to combine existing, published phylogenies into a single phylogeny. Here, we study supertree construction in the context of divide-and-conquer methods for large-scale tree reconstruction.

We study several divide-and-conquer approaches and experimentally demonstrate their advantage over Matrix Representation Parsimony (MRP), a traditional supertree technique, and over global heuristics such as the parsimony ratchet. On the ten large biological datasets under investigation, our study shows that the techniques used for dividing the dataset into subproblems as well as those used for merging them into a single solution strongly influence the quality of the supertree construction. In most cases, our merging technique—the Strict Consensus Merger (SCM)—outperforms MRP with respect to MP scores and running time. Divide-and-conquer techniques are also a highly competitive alternative to global heuristics such as the parsimony ratchet, especially on the more challenging datasets.

## 1 Introduction

Supertree methods combine smaller, overlapping subtrees into a larger tree. Their traditional application has been to combine existing, published phylogenies, on which the community agrees, into a tree leaf-labeled by the entire set of species. The most popular supertree method is Matrix Representation Parsimony (MRP) (Baum, 1992; Ragan, 1992), which has been used in a number of phylogenetic studies (Purvis, 1995;

---

\*Department of Computer Science, U. of Texas at Austin, [usman,tandy@cs.utexas.edu](mailto:usman,tandy@cs.utexas.edu)

<sup>†</sup>Department of Computer Science, U. of New Mexico, [moret,tlw@cs.unm.edu](mailto:moret,tlw@cs.unm.edu)

Bininda-Emonds et al., 1999; Bininda-Emonds and Sanderson, 2001; Liu et al., 2001; Jones et al., 2002). Bininda-Emonds and colleagues (Bininda-Emonds and Sanderson, 2001; Bininda-Emonds, 2003a) have evaluated the behavior of several variants of MRP on small simulated datasets with respect to topological accuracy.

We study the application of supertree methods in a different context: as part of divide-and-conquer methods that can be used to solve difficult optimization problems such as Maximum Parsimony (MP) and Maximum Likelihood (ML) (Felsenstein, 1981; Foulds and Graham, 1982; Steel, 1994; Hillis et al., 1996). These two problems are sufficiently hard that a biologically acceptable phylogenetic analysis can take a very long time (months, perhaps) to derive. The conjecture we study in this paper is that divide-and-conquer strategies can speed up searches for optimal trees under MP and ML.

A divide-and-conquer method for phylogeny reconstruction operates as follows.

- Step 1: Decompose the dataset into smaller, overlapping subsets.
- Step 2: Construct phylogenetic trees on the subsets using the desired “base” phylogenetic reconstruction method.
- Step 3: Merge the subtrees into a single (not necessarily fully resolved) tree on the entire dataset.
- Step 4: Refine the resulting tree to produce a binary tree.

Several divide-and-conquer methods have been developed and studied, including quartet-based methods, of which Quartet Puzzling (Strimmer and von Haeseler, 1996) is the most popular, and the family of Disk-Covering Methods (DCMs) (Huson et al., 1999a,b; Nakhleh et al., 2001; Warnow et al., 2001; Tang and Moret, 2003). In each of these methods, a supertree method (Step 3) is used to combine subtrees into a tree on the entire dataset. Supertree methods are thus an integral aspect of a divide-and-conquer strategy, but the other three aspects of such a strategy also affect accuracy and speed. Our study addresses the following questions:

- Should the subtrees used in reconstruction be carefully selected in terms of the subsets they represent or can the subsets be arbitrary as long as some overlap exists among them?
- Given a fixed collection of overlapping subtrees, what is the best method to assemble them into a single supertree?
- How do divide-and-conquer methods fare when compared to “global” approaches, such as the heuristic MP searches in PAUP\* (Swofford, 2002)?

To investigate the first two questions, we compare methods that differ explicitly in how they decompose the dataset and how they merge subtrees into a supertree. We consider two variants in the DCM family (named DCM1 and DCM2), plus (as a control) random decompositions; these decompositions are coupled with MRP and/or the Strict Consensus Merger (the supertree method developed for the DCM family) to merge the resulting subtrees into a single supertree; finally, all combinations of methods are followed by a refinement phase. To ascertain whether divide-and-conquer approaches can

outperform “global” approaches to solving MP or ML, we compare the performance of our DCM strategies with the parsimony ratchet (Nixon, 1999), one of the best performing MP heuristics for large datasets.

## 1.1 Overview of Experimental Results

We compare these methods on ten biological datasets that range from 328 to 854 taxa, focusing on the question of how techniques used for dataset decomposition and supertree reconstruction impact the running time and the MP score of the result. We find that the DCM2+SCM method outperforms the other methods on all our datasets. The specific decomposition technique has a significant impact on the MP score of the resulting tree as well as on running time, with DCM2 clearly outperforming random decompositions. Furthermore, we obtain improved MP scores in all decomposition strategies (DCM and random) when the subproblems are large—an observation that impacts taxon-sampling strategies. The supertree method used to combine subtrees into a single tree on the full dataset is also very important. When MRP and SCM are followed by the same resolution technique in Step 4, SCM generally produces better MP scores than MRP. The only exception was for DCM1-based decompositions, but these decompositions are relatively poor and not competitive (as our results show).

Our study demonstrates that the benefit of a divide-and-conquer technique depends on the properties of the dataset. When the dataset can be decomposed well by DCM2—into significantly smaller subproblems with good overlap, DCM2 provides a clear advantage (in running time or MP scores, as desired). The advantage is most pronounced for challenging datasets, datasets for which heuristic MP searches take a long time to find a first good solution. We compared DCM2-based approaches with the parsimony ratchet—the best MP global heuristic in our experiments—on two biological datasets: the well-studied 500 *rbcL* DNA dataset (Rice et al., 1997) and a set of 816 Bacterial rRNA sequences (Wuyts et al., 2002). The *rbcL* dataset decomposes poorly and is not especially challenging for MP heuristics; our study shows that DCM2 provide no improvement over the parsimony ratchet for this problem. In contrast, the rRNA dataset is quite challenging for MP heuristics but decomposes well; our study shows that DCM2 clearly improves on the parsimony ratchet for this problem. (Interestingly, DCM2-Ratchet, using the parsimony ratchet as a base method in a DCM2 decomposition, is almost as good as a global ratchet on the *rbcL* dataset, in spite of the very poor decomposition.)

## 1.2 Comparison with Previous Work

Bininda-Emonds and colleagues (Bininda-Emonds and Sanderson, 2001; Bininda-Emonds, 2003a) studied supertree reconstruction from an experimental point of view, focusing on the MRP method and using small simulated datasets. While we also study MRP, our focus is as much on decomposition as it is on supertree reconstruction and so we study several other methods; moreover, our testing uses biological datasets rather than simulated ones, thereby forcing us to use MP scores as our measure of accuracy (since the true trees for these datasets are not known); finally, we focus on large datasets (limited in this study to datasets below 1,000 taxa due to the dearth of larger published

datasets), since these are the datasets where a divide-and-conquer methodology will have the largest impact.

Some of the earliest divide-and-conquer methods are quartet-based methods, such as Quartet Puzzling (Strimmer and von Haeseler, 1996), Short Quartet methods (Erdős et al., 1997), and Quartet Cleaning (Berry et al., 1999). Quartet methods are at one extreme of divide-and-conquer methods, since they decompose the datasets into the smallest possible subsets for which nontrivial trees exist—subsets of just four taxa each. Quartet-based methods cannot profitably use either MRP or SCM (the two supertree reconstruction techniques we study here): MRP is too expensive given the tiny trees and SCM will usually return a totally unresolved tree because too many quartets will be in conflict. In an earlier study (St. John et al., 2001), we compared various quartet-based methods and the fast and simple neighbor-joining method (NJ) (Saitou and Nei, 1987) on simulated data. Quartet Puzzling, which merges quartet trees using a greedy heuristic, clearly dominated the other quartet-based methods, but was much slower and clearly less accurate than NJ. These results suggest that decompositions into tiny subsets is not profitable. Other published divide-and-conquer methods include Compartmentalization (Mishler, 1994), which is not fully described and so cannot be implemented, and a strategy used to analyze a biological dataset (Olsen et al., 1994), where again the decomposition and merging steps are not well enough described to enable one to implement and test the strategy.

## 2 Divide-and-Conquer Reconstruction Methods

Recall that a divide-and-conquer method uses four basic steps to construct a supertree from a given dataset,  $S$ :

- Step 1: Decompose the dataset into smaller, overlapping subsets.
- Step 2: Construct phylogenetic trees on the subsets using the desired “base” phylogenetic reconstruction method.
- Step 3: Merge the subtrees into a single (not necessarily fully resolved) tree on the entire dataset.
- Step 4: Refine the resulting tree to produce a binary tree.

Steps 2 and 4 are the same in all of our algorithms (except for our study of global heuristics versus divide-and-conquer methods in Section 5). We use a slow heuristic search for MP as the “base method” to construct the subtrees, but a fast heuristic search for MP to refine the merged supertree into a binary tree. Thus, our methods differ only in how they implement Steps 1 and 3. Sections 2.1 and 2.3 describe the techniques used for data decomposition and subtree merging. A summary of all of the supertree methods used in our study is given in Section 2.5.

## 2.1 Data Decomposition

### 2.1.1 DCM-based decomposition

Disk-Covering Methods (DCMs) (Huson et al., 1999a,b; Nakhleh et al., 2001; Warnow et al., 2001) are meta-methods for phylogenetic reconstruction: they operate in conjunction with a “base method” such as an MP heuristic or NJ. DCMs decompose the input set into smaller overlapping sets on which subtrees are computed using the specified base method. They have a dual goal: improved accuracy and better speed. Because the subsets have smaller diameter (maximum pairwise distance) than the original dataset, they are less likely to cause accuracy problems; and because the possibly expensive base methods only have to solve small subsets, the overall algorithm runs faster. One goal can be stressed at the expense of the other; thus there are several DCMs, each of which was designed for use with a particular base method.

The first DCM, DCM1 (Huson et al., 1999a), was designed for methods such as NJ, whose topological accuracy is negatively affected by large pairwise distances. DCM1 thus attempts to minimize the evolutionary diameter of each subproblem: it produces many subproblems, each with small diameter, but does not control the overlap between the subproblems. Earlier studies we conducted (and confirmed here) showed that DCM1 does not work particularly well with heuristic MP as a base method. Therefore, we developed DCM2 (Huson et al., 1999b), which produces a small number (two or three is typical in experiments presented here) of subproblems, all of which share one subset of taxa and are otherwise disjoint. Thus DCM2 tightly controls the overlap pattern, but does not directly attempt to control the diameter of each subset, thereby producing larger disks than DCM1.

The input to both DCM1 and DCM2 is a set  $S = \{s_1, \dots, s_n\}$  of  $n$  taxa (typically, aligned biomolecular sequences), an  $n \times n$  matrix,  $D = (d_{ij})$ , containing an estimate of the pairwise distances between the taxa, and a *threshold*—a particular  $q \in \{d_{ij}\}$ . Both methods start by computing a *threshold graph*,  $G(d, q)$ , defined as follows:

- The vertices of  $G(d, q)$  are the taxa,  $s_1, s_2, \dots, s_n$ .
- The edges of  $G(d, q)$  are those pairs  $(s_i, s_j)$  obeying  $d_{ij} \leq q$ .

The graph is then minimally *triangulated*, i.e., edges are added to the graph until every cycle of length at least four has a chord (an edge connecting two non-consecutive vertices on the cycle) (Buneman, 1974; Golubic, 1980), while attempting to minimize the weight of the largest edge added. Obtaining an optimal triangulation of a graph is in general NP-hard (Bodlaender et al., 1992), but threshold graphs are usually triangulated or close to it (Huson et al., 1999a)—and our experience shows that even the simple greedy heuristic produces triangulations that do not have very long edges. We triangulate the threshold graph because triangulated graphs have many computationally useful properties, notably:

- they have a linear number of maximal cliques (cliques that cannot be increased by adding a vertex) and these cliques can be computed in polynomial time; and
- their minimal vertex separators (minimal connected subgraphs whose removal breaks the graphs into disconnected pieces) are maximal cliques.

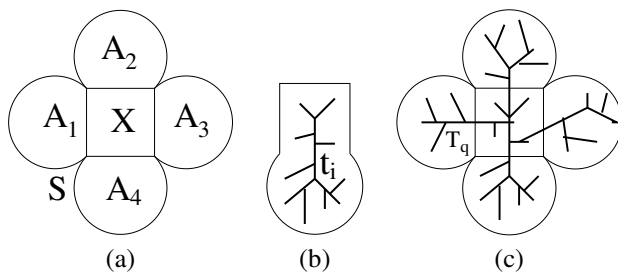


Figure 1: The three steps of Phase I in DCM2: (a) compute clique separator  $X$  for set  $S$  in threshold graph  $G(d, q)$ , producing subproblems  $A_1 \cup X, A_2 \cup X, \dots, A_r \cup X$ ; (b) compute tree  $t_i$  for each subproblem  $A_i \cup X$ ; and (c) merge computed subtrees to obtain tree  $T_q$  for set  $S$ .

(In contrast, these two problems are NP-hard for general graphs.) Thus, the next step in DCMs is to compute the maximal cliques. At this point, DCM1 is done and simply returns these cliques as the subproblems in the decomposition; these cliques have low diameter by construction. DCM2 scans through the cliques to find one clique  $X$  that minimizes  $\max_i |X \cup A_i|$ , where the  $A_i$  are the pieces into which the graph is broken upon removal of  $X$ ; it then returns the subsets  $X \cup A_i$  as the subproblems in the decomposition. Note that these subproblems have a unique common intersection, but that their diameter can be much larger (because of the addition of the separator  $X$ ) than that of a subproblem generated by DCM1. Figure 1 shows a symbolic representation of the DCM2 decomposition. We have proved that, as long as the subtrees are correctly inferred and the subproblems are large enough, the Strict Consensus Merger (SCM) technique applied to the subtrees will produce the true tree (Huson et al., 1999a). These theorems have ramifications for both DCM1- and DCM2-based strategies, but especially for DCM1 combined with distance-based methods; for these combinations it is possible to prove nice theorems about the sequence length requirements of the resulting methods (Huson et al., 1999a; Warnow et al., 2001).

However, our goal in this study is practical rather than theoretical: we want to develop faster and more accurate algorithms that perform well in practice. We experiment with different decompositions in order to determine which ones produce the best empirical results, so that improved MP scores are obtained faster. We pick a minimum triangulation to avoid grouping taxa that are evolutionarily distant (which would result in long edges in the triangulated graph). In developing the threshold graph, we need to choose a threshold  $q$ . The smallest useful value for  $q$  is  $d_0$ , the smallest possible value for which the threshold graph  $G(d, q)$  is connected; the largest possible value is simply  $\max\{d_{ij}\}$ . (Note that, if we applied the algorithm to this largest value, we would not obtain any decomposition into smaller subproblems, since the threshold graph would already be a clique.) In our experiments we look at ten equally spaced values between  $d_0$  and  $d_{10} = \max\{d_{ij}\}$  and run all tests with values  $d_0$  and  $d_4$ .

### 2.1.2 Random decomposition

As a control for DCM2, we also considered the effects of decomposing a dataset into random overlapping subsets, using three parameters: the number  $x$  of subproblems, the desired minimum size  $y$  of each subproblem, and the desired minimum size  $z$  of the pairwise intersection of subsets. Let  $n$  be the number of taxa to be distributed among the subsets. The  $x$  subsets are populated as follows. First,  $z$  taxa are randomly selected and all of them are placed into each of the subsets. For each subset, we then randomly select an additional  $y - z$  taxa from the remaining available taxa (marking the chosen  $y - z$  taxa as unavailable). Finally, if any taxa have not yet been placed in any particular subset, we add these taxa randomly to subsets. The resulting decomposition mimics the structure of DCM2 in that it produces subsets with a shared subset, but otherwise pairwise disjoint.

## 2.2 Base Methods: Heuristic Searches for MP

Heuristic searches for MP trees form a basic part of our divide-and-conquer reconstructions in three places: using a base method on subproblems to construct subtrees, using MRP to merge subtrees into a supertree, and refining the resulting tree into a binary tree. The heuristic MP search (HS) of PAUP\*4.0b10 (Swofford, 2002) was used for these analyses since the datasets are too large (in the hundreds) for exact optimization. Experiments were performed on simulated data in order to determine the quality of the HS needed in each stage.

- **Fast HS:** A fast heuristic search in which we save only one tree, starting from one initial random sequence addition ordering. We use the PAUP\*4.0b10 commands:  

```
set criterion=parsimony maxtrees=1 increase=no;
hsearch start=stepwise addseq=random swap=tbr hold=1 nreps=1;
```
- **Medium HS:** Medium heuristic search with ten random sequence addition orderings and 100 saved trees. We use the PAUP\*4.0b10 commands:  

```
set criterion=parsimony maxtrees=100 increase=no;
hsearch start=stepwise addseq=random swap=tbr hold=1 nreps=10;
contree all/ strict=yes;
```
- **Slow HS:** A slow heuristic search with 100 random sequence addition orderings and 1,000 saved trees. We use the PAUP\*4.0b10 commands:  

```
set criterion=parsimony maxtrees=100 increase=no;
hsearch start=stepwise addseq=random nreps=100 nchuck=1 chuckscore=1
swap=tbr;
set maxtrees=1000 increase=no;
filter best=yes;
hsearch start=current swap=tbr hold=1 nchuck=1000 timelimit=3600;
contree all/ strict=yes;
```

We also used the parsimony ratchet (Nixon, 1999) in a PAUP\*4.0b10 implementation written by Bininda-Emonds (Bininda-Emonds, 2003b). The ratchet is a simple and effective heuristic for general optimizing search and works iteratively as follows:

1. Run Fast HS for MP.
2. Randomly select 25% of the sites, set their weights to 2 and run Fast HS on the perturbed data, starting with the tree from the previous search.
3. Reset the site weights to their original values and run Fast HS starting with the tree from the previous search.
4. Repeat steps two and three as desired.

## 2.3 Merging Subtrees

### 2.3.1 Matrix representation parsimony (MRP)

The MRP approach encodes a set  $\mathcal{T}$  of trees as binary characters with missing values (i.e., “partial binary characters”) and then applies some heuristic for maximum parsimony on the resulting set of sequences. Understanding how MRP works thus requires understanding the encoding and the interpretation of partial binary characters.

Let  $S$  denote the full set of taxa and let  $T$  be one of the trees in the set  $\mathcal{T}$ —thus  $T$  has leaf set  $S_0 \subset S$ . Let  $e$  be an arbitrary edge in  $T$ . Deleting  $e$  from  $T$  partitions the leaves of  $T$  into two sets  $A$  and  $B$ . Now define a character  $c_e$  on all of  $S$  by setting

$$c_e(s) = \begin{cases} 0 & \text{if } s \in A \\ 1 & \text{if } s \in B \\ ? & \text{otherwise} \end{cases}$$

The set  $C(\mathcal{T}) = \{c_e : \exists T \in \mathcal{T}, e \in E(T)\}$  is the MRP encoding of the set  $\mathcal{T}$  of trees.

Given a set of sequences defined by partial binary characters and a candidate tree  $T$  on the set of sequences, all ?s are replaced by 0 or 1 in such a way as to minimize the parsimony score of the tree. If every subtree in the MRP analysis is accurate (i.e., topologically identical to the true tree induced on its set of leaves), then the true tree is one of the maximum parsimony trees. Hence, an exact solution to maximum parsimony will return the true tree as one of the solutions. (This observation follows from the fact that the true tree is a “perfect phylogeny” (Bodlaender et al., 1992) for the MRP-encoded set of sequences.)

For MRP, we used Slow HS. Since the search can identify more than one tree of lowest score, we return the strict consensus of the best trees found—the most resolved tree that is a common contraction of these best trees.

### 2.3.2 Strict-consensus merger (SCM)

The Strict Consensus Merger (SCM) combines a set of trees into a single tree. The merging is done pairwise until only one tree is left. The specific order in which the trees are merged matters when the subtrees are defined by a DCM1 decomposition, but is irrelevant when the subtrees are defined by a DCM2 decomposition; hence, for DCM2 it suffices to describe how SCM operates on two trees. (For specifics on how SCM operates in a DCM1 analysis, see (Huson et al., 1999a).)

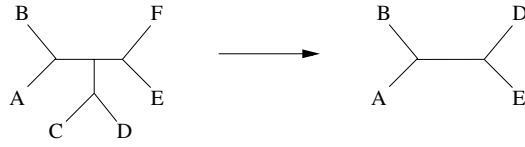


Figure 2: Tree  $T$  restricted to leaf set  $\{A, B, D, E\}$

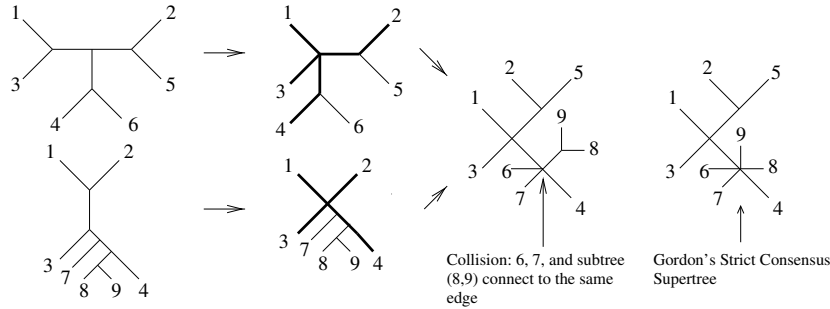


Figure 3: Handling collisions in the SCM and Gordon's Strict Consensus Supertree. The bipartition  $\{\{1,2,3,4,5,6,7\},\{8,9\}\}$  is present in the supertree under SCM, but not in Gordon's Strict Consensus Supertree.

Let  $L(T)$  denote the set of leaves of  $T$ ,  $C(T)$  denote the set of bipartitions of  $T$ , and  $T_X$ , with  $X \subseteq L(T)$ , denote the tree obtained by restricting the leaf set of  $T$  to  $X$  and suppressing nodes of degree 2 (see Figure 2). SCM takes two trees  $T_1$  and  $T_2$  and returns a tree  $T_{12}$  on the leaf set  $L(T_1) \cup L(T_2)$  according to the following procedure:

- Set  $X = L(T_1) \cap L(T_2)$ .  $X$  is the *backbone* and must satisfy  $|X| \geq 3$ .
- Compute the strict consensus,  $T_X$ , of  $T_1$  and  $T_2$ , each restricted to the leaf set  $X$ .
- Add the remaining taxa from  $T_1$  and  $T_2$  into  $T_X$  to form  $T$ , so as to preserve as much structure as possible. Some piece of each tree  $T_1$  and  $T_2$  may attach onto the same edge of  $T_X$  (causing a *collision*).

Figure 3 illustrates the SCM algorithm on incompatible trees with a collision, i.e., an edge in the backbone to which both trees contribute pieces; the backbone is highlighted with thick edges. The Strict Consensus Merger handles collisions in the following way. If an edge  $e$  of the backbone has a collision, then we subdivide the edge, producing a new node  $v_e$ , to which all contributions will be attached; in each subtree  $T$  contributing to this edge, we identify all pieces of  $T$  that should attach to that edge and attach them directly to  $v_e$ .

The Strict Consensus Merger of two trees is very similar to Gordon's Strict Consensus Supertree (SCS) (Gordon, 1986). When the trees are compatible (there is no collision), the two methods produce the same output. However, when there is a collision, the SCS tree can be a strict contraction of the SCM tree, because it may contract additional edges located within pieces involved in the collision.

## 2.4 Optimal Tree Refinement (OTR)

Merging subtrees into supertrees using MRP or SCM can result in unresolved trees; all steps up to and including the merging step are perhaps best seen as attempts to identify the best-supported edges. Resolving the remaining polytomies (by adding edges) so as to minimize the parsimony score of the resulting tree is the NP-hard *Optimal Tree Refinement (OTR)* problem (Bonet et al., 1998). To “solve” it, we pass unresolved trees as constraint trees to PAUP\*4.0b10 and use a fast MP heuristic search for a resolved tree, using the following command:

```
constraints c1 (monophyly) = <the unresolved tree which is used as constraint>;
set criterion=parsimony maxtrees=1 increase=no;
hsearch start=stepwise addseq=random swap=tbr hold=1 nreps=1
constraints=c1 enforce=yes;
```

## 2.5 Supertree Methods Studied

By varying the techniques used to obtain the dataset decomposition into subsets and those used to merge subtrees into supertrees, we obtain many different divide-and-conquer methods. For each such method, we also have a choice of parameters. We study DCM1 and DCM2, each with a supertree construction phase of MRP or SCM, plus the random decomposition followed by MRP. (SCM can be used to construct supertrees on an arbitrary set of subtrees, but the order in which the trees are merged can have a big impact on the resulting supertree; because of this, further research is needed before we can understand how SCM performs with random decompositions.) For DCM1, we use only threshold  $d_0$ , whereas for DCM2 we use both  $d_0$  and  $d_4$ . Thus the methods we test are:

- *DCM1 + SCM( $d_0$ )*
- *DCM2 + SCM( $d_0$ )*
- *DCM2 + SCM( $d_4$ )*
- *DCM1 + MRP( $d_0$ )*
- *DCM2 + MRP( $d_0$ )*
- *DCM2 + MRP( $d_4$ )*
- *RANDOM + MRP*

## 3 Experimental Methodology

We ran two sets of experiments. The first set of experiments was designed to test two conjectures: (i) that careful decomposition of the dataset is crucial to the success of supertree methods and that the DCM methods offer such a careful decomposition; and (ii) that the Strict Consensus Merger (SCM) developed as part of DCM is superior to MRP as a supertree assembly tool. The second set of experiments was designed to test our conjecture that divide-and-conquer methods are a competitive alternative to global heuristics.

We use large biological datasets to test these conjectures. Biological datasets suffer from several disadvantages when used in testing algorithms: (i) we cannot produce “tailored” biological datasets designed to test specific aspects of the reconstruction algorithms; (ii) we cannot judge the outcomes on the basis of accuracy (because we do not know the “true” tree) and so must instead rely on substitute criteria, such as maximum parsimony scores or maximum likelihood scores; and (iii) we cannot use them to predict behavior on other datasets (because we do not know how to relate the specific characteristics of one biological dataset to those of another). On the other hand, biological datasets offer data with all the biases and peculiarities that are so hard to produce in simulations. Thus the main use of “real-world” data is in spot-checking (Moret, 2002)—confirming that predictions made on the basis of simulation results hold for biological data or pinpointing problems with models when the datasets yield incompatible results. In this study, we choose biological datasets for two reasons: (i) we need them for spot-checking our conjectures, which we derived from large-scale simulation studies that we have already conducted (St. John et al., 2001; Nakhleh et al., 2002; Moret et al., 2002); and (ii) no comparable experimental study has been conducted—existing reports are limited to small biological datasets or to just one larger dataset.

Since our conjectures may hold in significant parts of the parameter space, but not everywhere, we study the effect of various parameter settings. We parameterize the decomposition in terms of subset sizes and mean coverage (where the mean coverage is the mean number of subsets in which a taxon appears). Of course, each taxon must appear in at least one subset, but reconstruction requires mean coverage greater than  $1 \times$  (otherwise we would obtain a forest and not a tree). We match the size and coverage characteristics of random decompositions to our DCM decompositions and study the variation in parsimony scores as a function of subset sizes or coverage.

### 3.1 The Datasets

We obtained ten biological datasets (all biomolecular sequences) from various sources. Below we give a brief description of each dataset, noting the number of sequences, their lengths, and the maximum p-distance (normalized Hamming distance) between any two sequences in the set.

1. A set of 328 ITS RNA sequences (946 sites) from the flowering plant *Asteracaeae* obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin; max p-distance = 0.524.
2. A set of 439 aligned rDNA sequences of Eukaryotes (2,461 sites) (Goloboff, 1999); max p-distance = 0.649.
3. A set of 476 aligned Metazoan DNA sequences (1,008 sites) (Goloboff, 1999); max p-distance = 0.445.
4. A set of 500 aligned *rbcL* DNA sequences (759 sites) (Rice et al., 1997); max p-distance = 0.334.
5. A set of 556 aligned 16S rRNA sequences (2,402 sites) for the Spirochaetes class of Bacteria (Maidak et al., 2001); max p-distance = 0.31.

6. A set of 567 “three-gene” (*rbcL*, *atpB*, and *18s*) aligned DNA sequences (4,592 sites) (Soltis et al., 2000); max p-distance = 0.15.
7. A set of 590 aligned small subunit Archaea rRNA sequences (1,962 sites) (Wuyts et al., 2002); max p-distance = 0.382.
8. A set of 695 aligned 16S rRNA sequences (2,550 sites) for the Cyanobacteria class of Bacteria (Maidak et al., 2001); max p-distance = 0.219.
9. A set of 816 aligned 16S rRNA sequences (1,253 sites) for the Bifidobacteriales (132), Acholeplasmataceae (234), and Flexibacteraceae(family) (450) families of Bacteria (Wuyts et al., 2002); max p-distance = 0.46.
10. A set of 854 aligned *rbcL* DNA sequences (937 sites) (Goloboff, 1999); max p-distance = 0.39.

Recall that the DCM-based approaches require a distance matrix to compute the threshold graph as the first step of its computation—and also that the distance matrix does not play any role in the phylogenetic reconstruction beyond this first step. We use the Kimura 2-parameter (plus Gamma) (Kimura, 1980; Yang, 1993) distance correction formula to compute a distance matrix for each dataset, using parameter values of  $\kappa = 2$  and  $\alpha = 1$  (the “default” values). We do not need the model to fit the data particularly well, since it affects only the choice of edges in the threshold graph; nevertheless, it is possible that a better distance correction (such as could be obtained using MODELTEST (Posada and Crandall, 1998)) would yield better results. In other words, our results with the DCM-based approaches should be regarded as pessimistic—they establish a lower bound, but are subject to further improvement.

## 3.2 Implementation and Platforms

Our DCM implementations are a combination of C++ (which uses LEDA 4.3) and Perl scripts; they were originally written by Daniel Huson and further expanded by us. The random decomposition is also a combination of C++ and Perl scripts and was written by us. To run the MP heuristics used for solving the subproblems, for MRP, and for OTR (optimal tree refinement), we use constrained search as implemented in PAUP\*4.0b10 (Swofford, 2002).

Our experiments were run on two sets of processors running Debian Linux: the phylofarm cluster of 9 dual 500MHz Pentium III processors, and 16 dual 733MHz Pentium III processors which are part of the 132-processor SCOUT cluster. For our running time analysis, we provide the running time (in seconds) of each of the four major steps separately, as follows:

- Decomposition: For the DCM-based methods this includes the running time for computing and triangulating the threshold graph and finding the subproblems. For the random methods it is the time to form the subproblems.
- Base method: This is the total running time for Slow HS on all the subproblems.
- Merge: This is the running time to merge the subtrees into a supertree using MRP or SCM.

- Optimal Tree Refinement (OTR): This is the cost of running Fast HS with the (unresolved) tree obtained in the previous step as a constraint tree in PAUP\*.

We only show selected data in the following sections; complete data from our experiments is available on our web site at [www.cs.utexas.edu/users/phylo/supertree\\_chapter/](http://www.cs.utexas.edu/users/phylo/supertree_chapter/).

## 4 Results on Decompositions

### 4.1 Comparing Different DCMs

We begin by examining the six different DCM-based approaches defined in Section 2.5. Figures 4 and 5 show relative MP scores and running times on the ten datasets. The best method (in terms of MP scores) is consistently DCM2+SCM at either  $d_0$  or  $d_4$ ; the other methods are not nearly as competitive. (MP scores that are lower even by these small percentages are often considered significant in phylogenetic analysis.)

Furthermore, SCM is better than MRP at combining subtrees in nearly all cases. (The only exception is DCM1 decompositions, but these decompositions are relatively poor and clearly not competitive.) Moreover, running times show that MRP is far more expensive than SCM—a matter of hours vs. seconds. Therefore, we focus on DCM2+SCM, since it is clearly the best divide-and-conquer strategy we tested. Our first task is to determine a suitable threshold. Figures 6 and 7 (using the data of Figures 4 and 5) show that DCM2+SCM( $d_4$ ) outperforms DCM2+SCM( $d_0$ ) on most of the ten datasets. This improvement in MP scores as we increase the threshold value is consistent with previous studies (Huson et al., 1999b) and our recent simulation studies (not shown). Note that, as we increase the threshold, the maximum subproblem size increases, but the number of subproblems decreases; hence the total running time may decrease. For DCM2+SCM, whether for threshold  $d_0$  or  $d_4$ , by far the most costly aspect of the reconstruction is the time spent in the MP heuristics—in reconstructing trees on the subsets and, to a lesser extent, in the OTR phase; in contrast, the DCM and SCM phases are very fast. (These data are not shown here, but are available from our web site.)

### 4.2 Comparing Random Decompositions

With random decompositions, we must use the MRP supertree method, since SCM is specifically designed for DCMs. Our goal here is to understand the effect of the (random) decomposition, in particular, the size of subsets and the amount of coverage, on the quality of reconstructions.

We want the coverage (the average number of subsets in which a taxon appears) to run from  $2\times$  to  $5\times$  and the size of the subproblems to range from 10% to 90% of the dataset, so we choose the number of subproblems to be

$$\text{number of subproblems} = \text{floor} \left( \frac{\text{coverage} \cdot \text{total size}}{\text{subproblem size}} \right)$$

So as not to bias the subset decomposition any further, we set the parameter  $z$  (the minimum overlap size) to 0 and let the pairwise overlap be induced through the number of subproblems and subproblem sizes as chosen above.

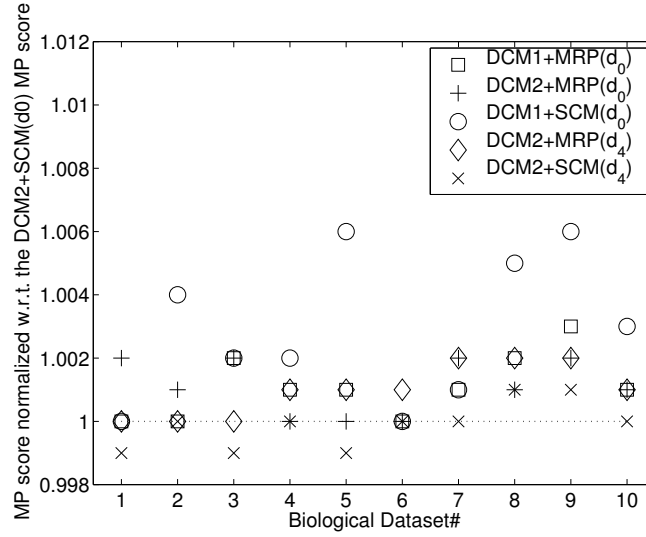


Figure 4: Comparison of the MP scores of DCM-based approaches on ten biological datasets, normalized with respect to the MP score of DCM2+SCM( $d_0$ ).

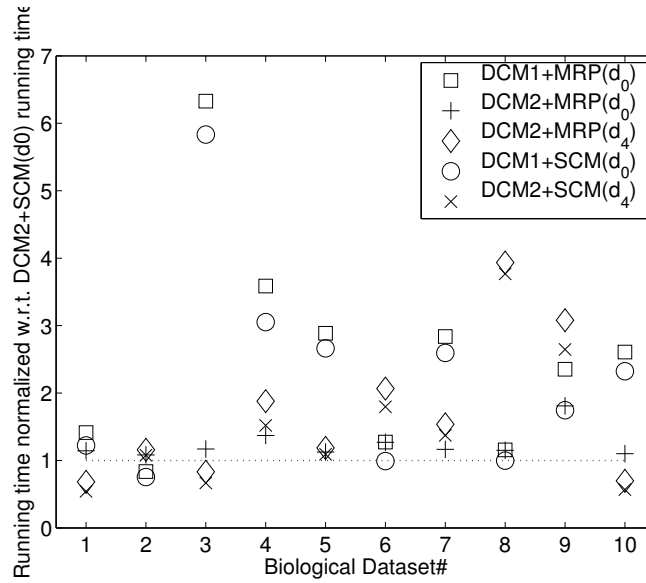


Figure 5: Comparison of the running times of DCM-based approaches on ten biological datasets, normalized with respect to the running time of DCM2+SCM( $d_0$ ).

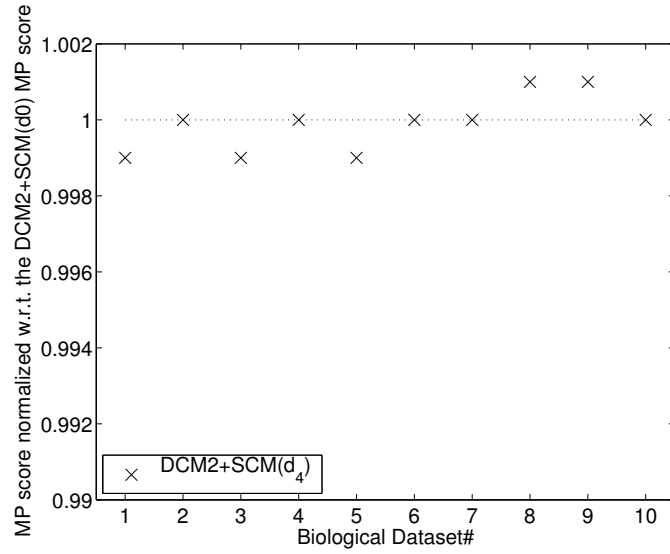


Figure 6: The ratio of the MP scores of DCM2+SCM( $d_0$ ) to those of DCM2+SCM( $d_4$ ) on ten biological datasets.

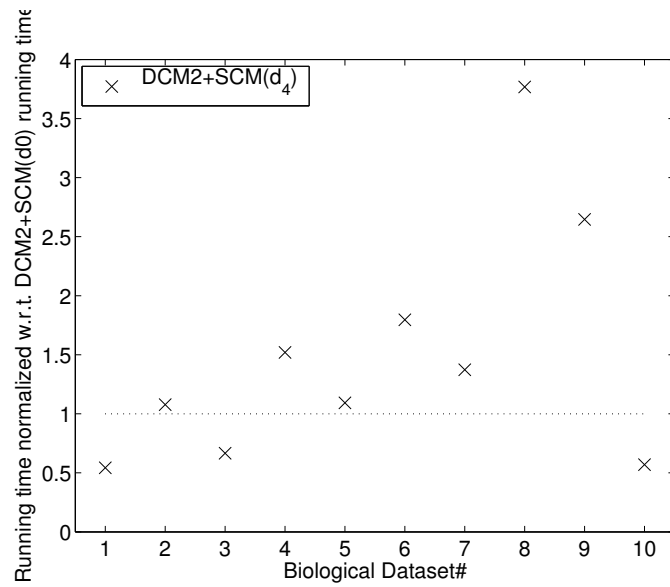


Figure 7: The ratio of the running times of DCM2+SCM( $d_0$ ) to those of DCM2+SCM( $d_4$ ) on ten biological datasets.

We solve the subproblems using Fast HS for MP (see Section 2) and run MRP using Medium HS for MP (defined by a time limit of 600 seconds). We examine 5 values of average subproblem sizes: 10%, 30%, 50%, 70% and 90% of the dataset. For each average subproblem size, we examine coverages of  $2\times$ ,  $3\times$ ,  $4\times$  and  $5\times$ . The detailed results on each of the 10 datasets are available on our website. Unsurprisingly, we find that MRP applied to random decomposition does much better with larger subsets and somewhat better with increased coverage, as was also observed in (Bininda-Emonds and Sanderson, 2001). Furthermore, as the subproblem sizes become larger, the MP scores of MRP on random decompositions slowly approach those of DCM2+SCM( $d_0$ ).

### 4.3 DCM vs. Random Decompositions

We now explore the relative performance of DCM2 decompositions and random decompositions. We run DCM2+SCM( $d_0$ ) only as a benchmark, but focus on DCM2+MRP( $d_0$ ), since we can ensure that MRP is applied to closely comparable decompositions. (We can set the three parameters for random decomposition so as to produce the same number of subsets as DCM2, with closely matched average subset sizes and coverage.) We use Slow HS for MP on the subproblems as well as for MRP and again report the average over 5 runs for the random decomposition. Figures 8 and 9 plot the ratios of MP scores (and running times) of DCM2+MRP and of MRP on random decompositions to the MP scores (and running times) of DCM2+SCM. (In addition MP scores for the trees obtained by each method, along with other details, can be found on our website.) The results clearly indicate that DCM2+SCM does better, in terms of both MP scores and running times, than either DCM2+MRP or MRP on random decompositions, with this last doing worst of all. Thus DCM2 decompositions are better than random decompositions and SCM does a better job at assembling supertrees from such decompositions than MRP (scores and resolution are both better). Moreover, MRP is very slow: on some datasets, the time difference is on the order of hours of computation, hours that could be used to conduct a more thorough parsimony search on the subtrees or in the OTR phase of DCM2+SCM. (We have not run such an equal-time comparison, but we expect that the gap in parsimony scores returned by DCM2+SCM and the other methods would be widened.)

## 5 Results on Global Heuristics

Our results suggest that a DCM2+SCM analysis is both faster and more accurate (in terms of MP scores) than the other divide-and-conquer methods studied. How then does DCM2+SCM compare to a direct (global) heuristic approach? We expect that the DCM approach will prove better on those datasets that yield good decompositions (into a small number of substantially smaller datasets with good overlap), but need to ascertain how the DCM approach performs when decompositions are poor.

We selected two datasets: the 500 *rbcL* dataset and the 816-taxon rRNA dataset. These two datasets are selected so as to explore how DCM2 behaves under extreme conditions. The first dataset is a poor candidate for improvement with DCM2: it decomposes poorly and is not challenging (simple heuristic searches quickly find a solu-

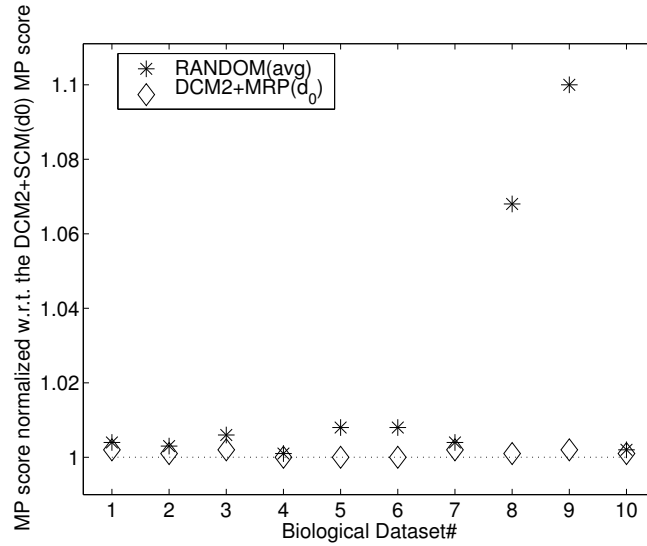


Figure 8: Comparison of MP scores of DCM-based methods and RANDOM (averaged over 5 runs) normalized with respect to the DCM2+SCM( $d_0$ ) MP scores on each of the ten biological datasets.

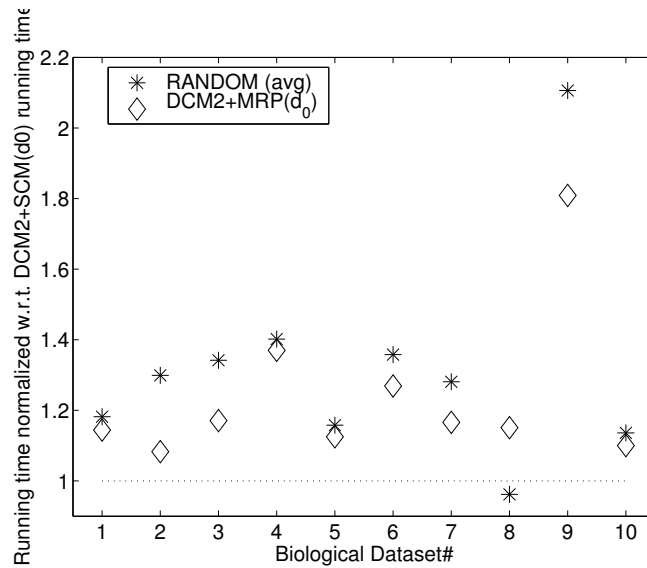


Figure 9: Comparison of running times of DCM-based methods and RANDOM (averaged over 5 runs) normalized with respect to the running time of DCM2+SCM( $d_0$ ), on each of the ten biological datasets.

tion within a couple of steps of the best score known). The second dataset, in contrast, should enable DCM2 to yield an improvement: it decomposes well and is quite challenging.

We first explore various global heuristics to identify the method that performs best on the the 500 *rbcl* dataset; this experiment shows that the parsimony ratchet significantly improves upon other local search heuristics in PAUP\*. We then use the parsimony ratchet both as a base method for DCM2+SCM (yielding a method we call DCM2-Ratchet, that also includes a final OTR phase) and as a global optimization heuristic, comparing the performance of these methods on each of our two datasets. Finally, we explore the performance of a two-phase technique in which we use DCM2+SCM to produce a starting tree for a subsequent search (using the ratchet) and we compare the performance of this two-phase technique to the ratchet. For this last experiment, we compare methods by examining the progress of each method over the period of time needed by the global parsimony ratchet to find the best score for each of the two datasets.

## 5.1 Local Improvement Heuristics on the 500 *rbcl* Dataset

We use the 500 *rbcl* dataset to explore the performance of various local improvement heuristics, as implemented in as implemented in PAUP\*4.0b10. These include the parsimony ratchet and heuristics of the form *fast-k max-m*. The fast-k max-m heuristic is implemented using the following commands:

```
set criterion=parsimony maxtrees=k increase=no;
hsearch start=stepwise addseq=random nreps=k swap=tbr nchuck=1 chuckscore=1;
set maxtrees=m increase=no;
hsearch start=current swap=tbr nchuck=m chuckscore=no;
```

We use 200 iterations of the ratchet (i.e., *ratchet200*) and vary *k* and *m*, thus producing the following set of heuristics to compare:

- fast100-max100
- fast500-max100
- fast1000-max100
- fast100-max1000
- fast500-max1000
- fast1000-max1000
- ratchet200

These heuristics are studied on the 500 *rbcl* dataset, restricted to just the parsimony-informative sites. The best score found to date on this dataset is 16,218. We address two questions: how quickly does each heuristic find the best known score and how quickly does it approach it?

We run each heuristic 10 times and collect the average MP score at each time step. Figure 10 shows the MP score of the best tree found by each heuristic as a function of time—up to a time beyond which none of the heuristics finds better trees. Note that

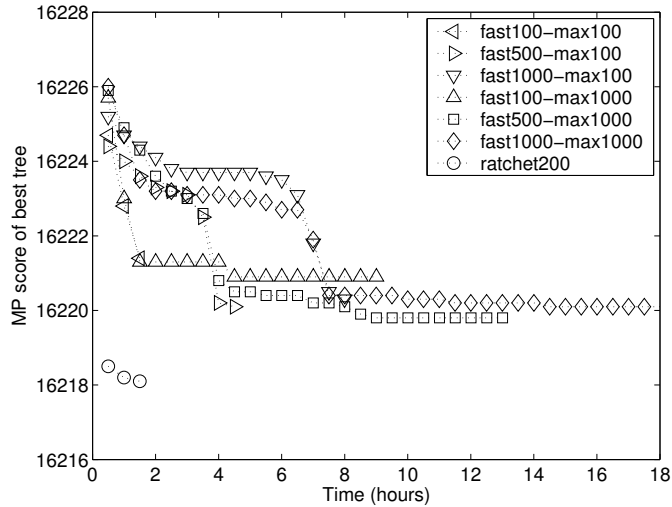


Figure 10: Comparison of heuristics on the *rbcL* dataset. Each heuristic was run 10 times; plotted is the average MP score at each time step.

the curve for ratchet200 lies strictly below the curves for the other methods: thus, at each time point, ratchet200 finds shorter trees than any of the other methods. Also, ratchet200 finds trees with the lowest known length within two hours, while the other methods cannot find such short trees. Thus, the ratchet is much more effective than the fast-k max-m techniques in finding trees with low MP scores. Other experiments (not shown here) confirm that the ratchet is better on our large datasets than the fast-k max-m heuristics. We therefore use the ratchet as the base method for DCM2 and compare it against the global ratchet.

## 5.2 Global Ratchet vs. DCM2-Ratchet on the 500 *rbcL* Dataset

We use DCM2-Ratchet with two different levels of OTR. We use the smallest possible threshold ( $d_0$ ) and obtain decompositions that minimize the maximum subproblem size. Unfortunately, even the smallest threshold yields a huge separator of size 411, so that the two subsets in the decomposition have sizes 455 and 456—quite a poor decomposition. (A larger threshold cannot help, since it would produce even larger subproblems.) We compare these two variants of DCM2-Ratchet against 200 iterations of the global ratchet. Each method is run 10 times and the average MP score at each time step is collected. Figure 11 shows that the DCM2-boosted variants of the ratchet are able to find trees almost as good as those found by the global ratchet, but not as fast.

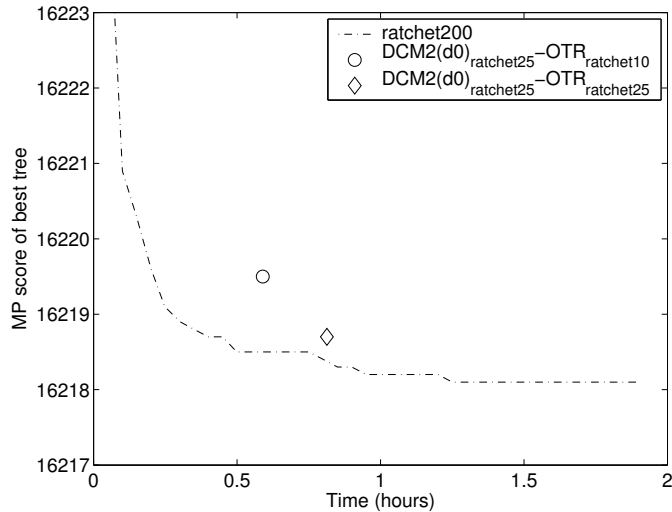


Figure 11: Comparison of DCM2-Ratchet to ratchet200 on Dataset #4 (averaged over 10 runs). The average subproblem size is 91%.

### 5.3 Global Ratchet vs. DCM2-Ratchet on the 816 rRNA Dataset

On the 816 rRNA dataset, the DCM2 decomposition at threshold  $d_0$  produces 2 subproblems of sizes 369 and 450—a good result. Unfortunately, the separator is tiny (just 3 nodes), which makes it difficult for SCM to merge trees with sufficient accuracy. Therefore, we pick a larger separator (with 36 nodes) in order to get more overlap. For this separator, we get three subproblems of sizes 132, 270, and 486.

Since this is a larger dataset than the 500 *rbcL*, we use 500 iterations of ratchet (ratchet500) for the global analysis. The subproblems are again computed using 25 iterations of the ratchet and three different iteration counts are used for the OTR phase: ratchet5, ratchet10, and ratchet25. Each method is run 5 times and the average MP score at each time step is collected. The global ratchet version finishes in about 48 hours. Figure 12 shows that, within the first two hours, DCM2 finds better trees than the global ratchet.

### 5.4 Using DCM2-Ratchet for Initialization

Because DCM2-Ratchet is fast and returns good solutions, it could prove useful in a two-phase optimization procedure, by providing strong initial solutions from which to start a global search. To study this approach, we run the global ratchet with the DCM2-Ratchet trees as starting trees; as might be expected, the global ratchet finds better trees when started with these initial solutions than when started from scratch. Figures 13(a) and 13(b) show the curve of (average) scores as a function of time for the global ratchet alone (started from scratch) and the global ratchet applied to the initial solutions returned by DCM2-Ratchet. For dataset #9 (the 816 RNA dataset), for

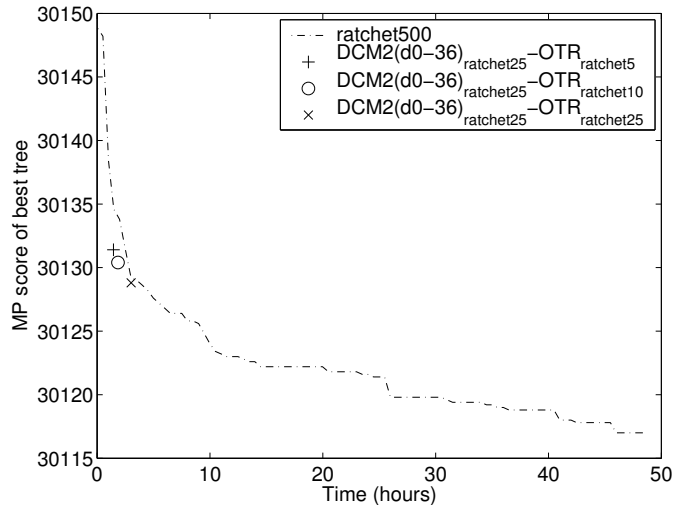


Figure 12: Comparison of DCM2-Ratchet to ratchet500 (best heuristic) on Dataset #9 (averaged over 5 runs). The average subproblem size is 36% (maximum is 60%) and the separator size is 4% of the problem size.

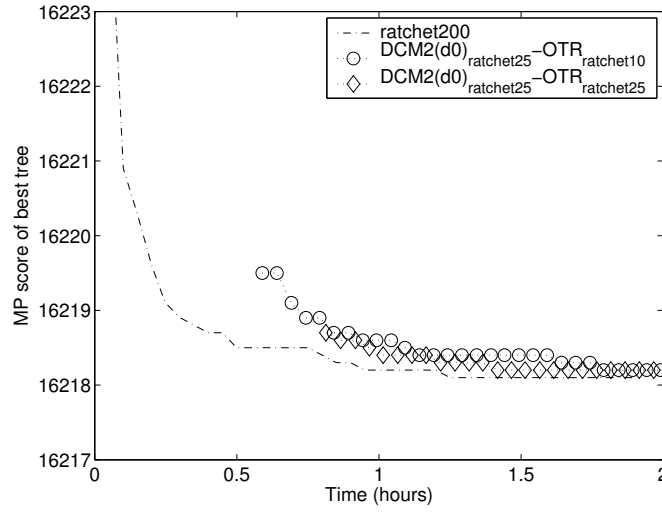
instance,  $\text{DCM2}_{\text{ratchet25-OTR}_{\text{ratchet25}}}$  finds trees with an MP score of 30,115 within approximately 26 hours, whereas the best trees found by the global ratchet (at the end of 48 hours) have an MP score of 30,117.

## 6 Summary and Conclusions

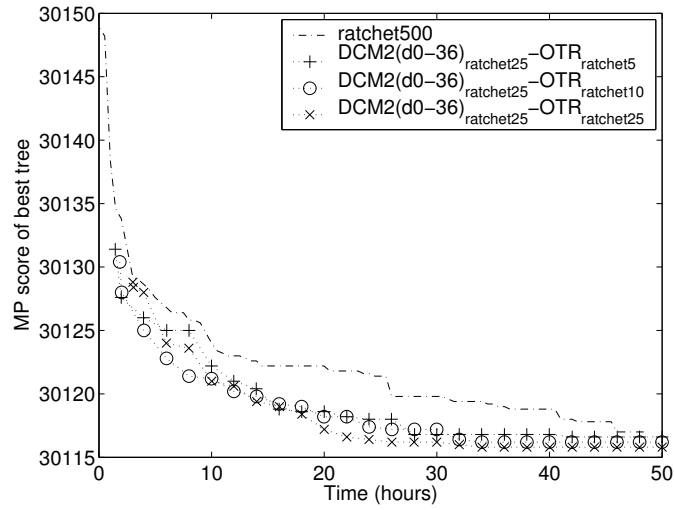
We set out to explore the potential of divide-and-conquer methods to improve the speed and accuracy of maximum parsimony searches; in particular, we wanted to learn which decomposition strategies and which supertree assembly techniques work well in such approaches. Our study confirms that divide-and-conquer methods can speed up searches for optimal MP trees, but (unsurprisingly) only when the decompositions are good.

The specifics of the divide-and-conquer strategy make a large difference. We had already shown that quartet-based methods (an extreme form of divide-and-conquer) are not competitive. We now find that random decompositions are clearly inferior to carefully crafted ones (e.g., decompositions obtained by DCM2) and that the strict consensus merger (SCM) technique for merging subtrees is both more accurate and much faster than the most commonly used supertree method, matrix representation parsimony (MRP). Both approaches, however, usually require an optimal tree-refinement (OTR) phase in which the supertree is refined into a binary tree, a phase to which more attention should be given in future work.

The significance of this study is both enhanced and limited by our use of biological datasets: we ensure relevance, but can only conduct fairly simple tests—a large



(a) Dataset #4 (*rbcL500*), using *ratchet200*



(b) Dataset #9 (*816 RNA*), using *ratchet500*

Figure 13: Comparison of the global ratchet started from scratch against the same started from the DCM2-Ratchet solutions.

simulation study is required to confirm our findings as well as discern more subtle effects. Other research questions suggested by this study include: (i) how best to decompose datasets for which DCM2 does not produce a good decomposition? (ii) how long should base methods be allowed to run on subproblems? (iii) what is the influence of the OTR phase on the entire process? and (iv) how can we best combine divide-and-conquer and global approaches (in the style of the approach discussed in Section 5.4)?

All of the work in this study concerns maximum parsimony, but divide-and-conquer methods, including the DCMs, are equally applicable to maximum likelihood—thus a study of DCM-ML approaches remains to be conducted.

Finally, MP and ML are surrogate optimization criteria for the real goal, which is topological accuracy (unmeasurable in absence of knowledge of the “truth”), hence conclusions about the accuracy of reconstruction must await a simulation study, in which the “true” trees are known. We conjecture that, while large decreases in parsimony scores (or large increases in likelihood scores) do translate into increased topological accuracy, small changes in these scores around near-optimal values have a nearly random effect on topological accuracy—in which case there is no point in spending additional days of computation to improve a score by 0.01% and every reason to devise early termination tests.

We conclude with a caveat: in reconstructing a very large tree, such as the Tree of Life with millions of taxa, we may not have the luxury of choosing our decompositions—the data-gathering process may have made that choice for us, at least at the higher taxonomic levels. For instance, significant data may be missing for many taxa, so that it is not feasible to analyze all sequences all at once. In such a case, the dataset decomposition will be given to us and thus will not be adjustable (except for the breaking of large clusters). We thus also need a large-scale evaluation of supertree methods in their traditional use.

## 7 Acknowledgments

This work was supported by the National Science Foundation under grants ACI 00-81404 (Moret), DEB 01-20709 (Moret and Warnow), EIA 01-13095 (Moret), EIA 01-13654 (Warnow), EIA 01-21377 (Moret), EIA 01-21680 (Warnow), and EIA 99-85991 (Warnow, the SCOUT Cluster), by the David and Lucile Packard Foundation (Warnow), and by an Alfred P. Sloan Foundation Postdoctoral Fellowship in Computational Molecular Biology, U.S. Department of Energy DE-FG03-02ER63426 (Williams).

## References

- Baum, B. (1992). Combining trees as a way of combining data sets for phylogenetic inference, and the desirability for combining phylogenetic trees. *Taxon* 41, 3–10.
- Berry, V., T. Jiang, P. E. Kearney, M. Li, and T. Wareham (1999). Quartet cleaning: Improved algorithms and simulation. In *Proc. 7th European Symposium on Algorithms (ESA 99)*, pp. 313–324. Springer Verlag. Volume 1643 of LNCS.

- Bininda-Emonds, O. (2003a). MRP supertree construction in the consensus setting. In M. Janowitz, F.-J. Lapointe, F. McMorris, B. Mirkin, and F. Roberts (Eds.), *Bioconsensus*, Volume 61 of *DIMACS Monographs*, pp. 231–242. American Mathematical Society.
- Bininda-Emonds, O. (2003b). Ratchet implementation in PAUP\*4.0b10. available from <http://www.tierzucht.tum.de:8080/WWW/Homepages/Bininda-Emonds>.
- Bininda-Emonds, O. and M. Sanderson (2001). Assessment of the accuracy of matrix representation with parsimony analysis supertree construction. *Systematic Biology* 50, 565–579.
- Bininda-Emonds, O. R. P., J. L. Gittleman, and A. Purvis (1999). Building trees by combining phylogenetic information: a complete phylogeny of the extant Carnivora (Mammalia). *Biological Reviews* 74, 143–175.
- Bodlaender, H., M. Fellows, and T. Warnow (1992). Two strikes against perfect phylogeny. In *Proc. Int'l Conf. Automata, Languages, and Programming ICALP'92*, Volume 623 of *Lecture Notes in Computer Science*, pp. 273–283. Springer-Verlag.
- Bonet, M. L., M. Steel, T. Warnow, and S. Yooseph (1998). Better methods for solving parsimony and compatibility. *Journal of Computational Biology* 5(3), 391–408.
- Buneman, P. (1974). A characterization of rigid circuit graphs. *Discrete Mathematics* 9, 205–212.
- Erdős, P. L., M. Steel, L. Székély, and T. Warnow (1997). A few logs suffice to build almost all trees— I. *Random Structures and Algorithms* 14, 153–184.
- Felsenstein, J. (1981). Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution* 17, 368–376.
- Foulds, L. R. and R. L. Graham (1982). The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics* 3, 43–49.
- Goloboff, P. (1999). Analyzing large data sets in reasonable times: solution for composite optima. *Cladistics* 15, 415–428.
- Golumbic, M. (1980). *Algorithmic graph theory and perfect graphs*. Academic Press Inc.
- Gordon, A. D. (1986). Consensus supertrees: the synthesis of rooted trees containing overlapping sets of leaves. *Journal of Classification* 3, 335–348.
- Hillis, D., C. Moritz, and B. Mable (1996). *Molecular Systematics*. Sinauer Pub., Boston.
- Huson, D., S. Nettles, and T. Warnow (1999a). Disk-covering, a fast-converging method for phylogenetic tree reconstruction. *Journal of Computational Biology* 6, 369–386.

- Huson, D., L. Vawter, and T. Warnow (1999b). Solving large scale phylogenetic problems using DCM2. In *Proc. 7th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB'99)*, pp. 118–129. AAAI Press.
- Jones, K. E., A. Purvis, A. MacLarnon, O. R. P. Bininda-Emonds, and N. B. Simmons (2002). A phylogenetic supertree of the bats (mammalia: Chiroptera). *Biological Reviews* 77(2), 223–259.
- Kimura, M. (1980). A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution* 16, 111–120.
- Liu, F.-G. R., M. M. Miyamoto, N. P. Freire, P. Ong, and M. Tennant (2001). Molecular and morphological supertrees for eutherian (placental) mammals. *Science* 291(5509), 1786–1789.
- Maidak, B., J. Cole, T. Lilburn, C. P. Jr, P. Saxman, R. Farris, G. Garrity, G. Olsen, T. Schmidt, and J. Tiedje (2001). The RDP-II (Ribosomal Database Project). *Nucleic Acids Research* 29(1), 173–4.
- Mishler, B. D. (1994). Cladistic analysis of molecular and morphological data. *American Journal of Physical Anthropology* 94, 143–156.
- Moret, B. (2002). Towards a discipline of experimental algorithmics. In M. Goldwasser, D. Johnson, and C. McGeoch (Eds.), *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, Volume 59 of *DIMACS Monographs*, pp. 197–213. American Mathematical Society.
- Moret, B., U. Roshan, and T. Warnow (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*, pp. 343–356. Springer-Verlag.
- Nakhleh, L., B. Moret, U. Roshan, K. S. John, and T. Warnow (2002). The accuracy of fast phylogenetic methods for large datasets. In *Proc. 7th Pacific Symp. Biocomputing (PSB'2002)*, pp. 211–222. World Scientific Pub.
- Nakhleh, L., U. Roshan, K. St. John, J. Sun, and T. Warnow (2001). Designing fast converging phylogenetic methods. In *Proc. 9th Int'l Conf. on Intelligent Systems for Molecular Biology (ISMB'01)*, Volume 17 of *Bioinformatics*, pp. S190–S198. Oxford U. Press.
- Nixon, K. C. (1999). The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics* 15, 407–414.
- Olsen, G. J., C. R. Woese, and R. Overbeek (1994). The winds of (evolutionary) change: breathing new life into microbiology. *Journal of Bacteriology* 176, 1–6.
- Posada, D. and K. A. Crandall (1998). Modeltest: testing the model of dna substitution. *Bioinformatics* 14(9), 817–818.

- Purvis, A. (1995). A composite estimate of primate phylogeny. *Philosophical Transactions of the Royal Society of London Series B* 348, 405–421.
- Ragan, M. (1992). Phylogenetic inference based on matrix representation of trees. *Molecular Phylogenetics and Evolution* 1, 53–58.
- Rice, K., M. Donoghue, and R. Olmstead (1997). Analyzing large datasets: *rbcL* 500 revisited. *Systematic Biology* 46(3), 554–563.
- Saitou, N. and M. Nei (1987). The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution* 4, 406–425.
- Soltis, D. E., P. S. Soltis, M. W. Chase, M. E. Mort, D. C. Albach, M. Zanis, V. Savolainen, W. H. Hahn, S. B. Hoot, M. F. Fay, M. Axtell, S. M. Swensen, L. M. Prince, W. J. Kress, K. C. Nixon, and J. S. Farris (2000). Angiosperm phylogeny inferred from 18s rDNA, *rbcL*, and *atpB* sequences. *Botanical Journal of the Linnean Society* 133, 381–461.
- St. John, K., T. Warnow, B. Moret, and L. Vawter (2001). Performance study of phylogenetic methods: (unweighted) quartet methods and neighbor-joining. In *Proc. 12th Ann. Symp. Discrete Algorithms (SODA'01)*, pp. 196–205. SIAM Press.
- Steel, M. A. (1994). The maximum likelihood point for a phylogenetic tree is not unique. *Systematic Biology* 43(4), 560–564.
- Strimmer, K. and A. von Haeseler (1996). Quartet puzzling: A quartet maximum likelihood method for reconstructing tree topologies. *Molecular Biology and Evolution* 13(7), 964–969.
- Swofford, D. L. (2002). PAUP\*: Phylogenetic analysis using parsimony (and other methods). Sinauer Associates, Underland, Massachusetts, Version 4.0.
- Tang, J. and B. Moret (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*, pp. i305–i312.
- Warnow, T., B. Moret, and K. St. John (2001). Absolute convergence: True trees from short sequences. In *Proc. 12th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'01)*, pp. 186–195. SIAM Press.
- Wuyts, J., Y. V. de Peer, T. Winkelmans, and R. D. Wachter (2002). The European database on small subunit ribosomal RNA. *Nucleic Acids Research* 30, 183–185.
- Yang, Z. (1993). Maximum likelihood estimation of phylogeny from DNA sequences when substitution rates differ over sites. *Molecular Biology and Evolution* 10, 1396–1401.