Rec-I-DCM3: A Fast Algorithmic Technique for Reconstructing Large Phylogenetic Trees

Usman Roshan * Bernard M.E. Moret† Tiffani L. Williams†
Tandy Warnow* ‡

Abstract

Estimations of phylogenetic trees are most commonly obtained through the use of heuristics for maximum parsimony (MP), an NP-hard problem. Although apparently good heuristics have been devised, even they fail to produce good solutions in reasonable time for large datasets—today's limit is unknown, but is probably less than one thousand sequences. In this paper, we present a promising new divide-and-conquer technique, *Recursive-Iterative-DCM3* (Rec-I-DCM3). This new method belongs to our family of Disk-Covering Methods (DCMs); it operates by iteratively dividing the input set of sequences into smaller overlapping subproblems, solving them using some base method (e.g., neighbor-joining, heuristic MP, heuristic ML, etc.), and then merging these subtrees into a single, phylogenetic tree. Thus, Rec-I-DCM3 is designed to *boost* the performance of its base method. Our new method is composed of a new DCM, which we call DCM3, but utilizes recursion and iteration as well; the result is a booster of phylogenetic reconstruction methods that can produce dramatic improvements for standard heuristics, as well as substantial improvements over the very best methods (which are harder to improve). We demonstrate the power of this new DCM on ten large biological datasets ranging from 1,322 to 13,921 sequences.

Contact: usman@cs.utexas.edu
URL: http://www.cs.utexas.edu/users/usman/ismb04
Keywords: Phylogeny, maximum parsimony, algorithms, heuristics

1 Introduction

Maximum parsimony (MP) and maximum likelihood (ML) are two of the major optimization problems in phylogeny (i.e., evolutionary tree) reconstruction. Both are quite hard to solve (MP is NP-hard [3], and ML harder in practice), and datasets above 30 taxa cannot be solved exactly with any reliability (branch and bound, or exhaustive search, techniques are both limited to smaller datasets). ML heuristics have been used to analyze datasets of up to a hundred or so taxa, but larger

*Department of Computer Science, U. of Texas at Austin, usman,tandy@cs.utexas.edu
†Department of Computer Science, U. of New Mexico, moret,tlw@cs.unm.edu
‡Radcliffe Institute for Advanced Study twarnow@radcliffe.edu

datasets seem beyond the capabilities of existing ML heuristics. MP heuristics are the main way by which systematists perform their phylogeny reconstructions (Sanderson surveyed 882 phylogenetic analyses published in 76 journals, and observed that 60% of the phylogenies were constructed using MP heuristics [14]).

Because of the importance of MP analyses in phylogeny reconstruction, systematists and algorithms researchers in phylogeny have studied the existing methods (specifically, implementations of heuristics in different software packages) to see which performed the best. The main criterion by which these methods have been studied is the time needed to get to the optimal score (or, more accurately, the best known score) on various real datasets, preferably of at least one hundred sequences. These studies [4, 11, 15, 12, 13] have suggested that the "parsimony ratchet" [11] was more effective than Tree-Bisection and Reconnection (TBR) hill-climbing [9], and that TNT's [4] implementation of the parsimony ratchet was more efficient than PAUP*'s [16] implementation. Thus, TNT's ratchet is probably among the best of the existing software tools for solving MP.

In an independent development, new algorithmic strategies were also developed which were hoped might result in improved parsimony heuristics. Divide-and-conquer methods in particular were thought to be promising. The motivation behind this approach was that an exponential time method (such as a thorough MP analysis would involve) would be faster if run on a small number of datasets, each a fraction of the size of the full dataset.

An early such divide-and-conquer method called DCM2 was developed for MP analysis, and presented at ISMB 1999 [7]. This divide-and-conquer method is used with an existing "base method" as follows. First a decomposition of the set of taxa into overlapping subsets is obtained, and trees are constructed on the subsets using the base method. Then the trees on the different subsets are merged together into a tree on the full dataset. The study in [7] in which the DCM2 technique was presented suggested that this divide-and-conquer approach might speed up maximum parsimony analyses, but the study was limited to performance on simulated data, and used the default heuristic within PAUP*, which is based solely on TBR hill-climbing.

Over the last two years we have attempted to evaluate the performance of the DCM2 technique. Our studies (briefly reported upon in [13], and extended here) were disappointing. While we saw that DCM2 sped-up the default (TBR-based) PAUP* heuristic for MP on real datasets, it failed to provide advantages on most datasets when we used more effective techniques (such as the parsimony ratchet) as the base method. This observation is consistent with the conjecture we had that when the base method is not that efficient at finding good solutions, DCM2 can provide a dramatic advantage, but as the base method improves in speed towards near-optimal solutions, DCM2's ability to boost the performance lessens. Consequently, DCM2's ability to consistently boost the performance of TBR hill-climbing but inconsistent performance with the parsimony ratchet made sense. We therefore needed to improve the design of the DCM.

The problem with DCM2's design seemed to be its decomposition: firstly, it turned out to be costly (even though polynomial time) to compute the DCM2 decomposition, and secondly, the decomposition it obtained often did not significantly decrease the size of the subproblems from the full dataset size. We then turned to developing new divide-and-conquer techniques which would be faster to compute, and also more likely to produce better decompositions. Our first attempt produced the DCM3 technique (or third Disk-Covering Method). As hoped, the DCM3 decompo-

sition is faster to compute than the DCM2 decomposition, and produces about the same number of subproblems as DCM2, but much smaller ones. However, when used with TNT's ratchet for the base method, DCM3 does not always provide improvements in MP analyses. We therefore turned to other algorithmic techniques: recursive use of DCM3 (so that we further subdivide subproblems), and iterative use (so that we periodically call DCM3 during the MP analysis). Our study, presented here, shows how these various methods perform in MP analyses of ten large real biological datasets.

The main contribution of the paper is the presentation of these new heuristics, none of which has been published before, and the empirical study evaluating their performance. As expected, the best performing method, *Recursive-Iterative-DCM3*, or *Rec-I-DCM3*, is very powerful: we show that when used with TNT's ratchet on subproblems, it produces significant speed-ups over TNT's ratchet in terms of the time to optimal. More significantly, we show that at every point in time during a 24 hour analysis, our Rec-I-DCM3 obtains better MP scores than TNT's ratchet.

2 Maximum Parsimony

We now formally define the Maximum Parsimony (MP) problem. Let S be a set of n sequences over a fixed alphabet Σ, all of the same length. Let T be a tree leaf-labelled by the set S and with internal nodes labelled by sequences over Σ of the same length. The *length* of T with this labelling is the sum, over all the edges, of the Hamming distances between the labels at the endpoints of the edge. (The Hamming distance between two strings of the same length is the number of positions in which they differ.) The MP problem seeks the tree T leaf-labelled by S with the minimum length; this is the same as seeking the tree with the smallest number of point mutations for the data. While MP is NP-hard [3], constructing the optimal labeling of the internal nodes of a fixed tree T can be done in polynomial time [2].

Iterative Improvement Methods: Iterative improvement methods are some of the most popular methods in phylogeny reconstruction. Some fast technique is used to find an initial tree; that tree is then improved through a local search, in order to find a different tree with a better score. The most popular local move is called Tree-Bisection and Reconnection, or TBR [9].

The Parsimony Ratchet: The parsimony ratchet is a technique that combines TBR hill-climbing with a specific technique in order to move out of local optima. Precisely, when the TBR hill-climbing reaches a local optimum, the ratchet modifies the input data (by doubling a random subset of one fourth of the sites, thus producing a set of sequences that are 1.25 times as long as the input sequences), and running TBR hill-climbing on the new data. When that new search reaches a local optima, then the dataset is changed back to the original dataset, and hill-climbing is resumed.

Disk-Covering Methods: Disk-Covering Methods (DCMs) [6, 7, 10, 13, 17] are divide-and-conquer methods that are designed to "boost" the performance of phylogenetic reconstruction methods. The first DCM [6], also called DCM1, was designed for use with distance-based methods and has provable theoretical guarantees about the sequence length required to reconstruct the true tree with high probability under Markov models of evolution (see [17]). The second DCM, DCM2 [7], was designed to speed up heuristic searches for MP trees; we showed [7] that, when DCM2 was used with PAUP* [16] TBR search, it produced better trees faster on simulated datasets. We also observed that the parsimony ratchet in PAUP* was better than TBR search at solving MP

[13] and so compared DCM2(PAUP*-ratchet) with the PAUP*-ratchet on biological datasets. We found varying results: sometimes DCM2(PAUP*-ratchet) was better, and sometimes worse, than the PAUP*-ratchet at solving MP (unpublished data, but see also [13]).

3 The DCM3 technique and its variants

DCM2's drawbacks are that (1) it takes a long time to compute a decomposition and (2) when it does so, it usually (on most datasets) contains subproblems almost as large as the original dataset. These two observations led us to design a new DCM, which we call "DCM3", for the Third DCM. The main change in the decomposition strategy is that DCM2 operates on the basis of an estimated distance matrix on the input, whereas DCM3's decomposition is obtained on the basis of a "guide tree" (which can change during the course of the heuristic search) for the dataset. Consequently, DCM3 can be used at different times to obtain different decompositions, thus enabling an iterative use of the decomposition strategy. Our experiments (reported on in this paper) show that DCM3 reliably produces a small number of subproblems, each of which contains at most 40% or 50% of the taxa. Also, whereas in our DCM2 strategy we explicitly minimize the size of the largest subproblems during the decomposition, we use a very fast heuristic to decompose the dataset (without trying to explicitly minimize the largest subproblem); this allows us to get good decompositions much faster than we could with DCM2. In the next sections, we describe the DCM3 decomposition and four methods based upon it: DCM3, Recursive-DCM3, Iterative-DCM3, and Recursive-Iterative-DCM3.

3.1 DCM3 decompositions

We begin by describing the DCM3 decomposition. We assume we have a tree T on our set S of taxa, and an edge weighting w of T (i.e., $w : E(T) \rightarrow \Re^+$). Most typically this edge-weighting will be given by the Hamming distances under the maximum parsimony labelling of the nodes of T (computable in polynomial time). Based upon this edge-weighted tree, we obtain a decomposition of the leaf set using the following steps. We begin by constructing the *short subtree graph*, which is the union of cliques formed on "short subtrees" around each edge.

Short subtrees of edges: Let e be an edge in a guide tree T, with edge weighting w. A *short quartet* around e is composed of four leaves (one from each of these four subtrees), where each leaf is selected to be closest to the edge e (the distance between nodes u and v is measured as $\sum_{e \in P_{uv}} w(e)$). Let $X(e)$ be the set of all leaves that are elements in a short quartet around e; we call $X(e)$ the "short subtree" around e. The graph formed by taking the union of all the cliques on all $X(e)$'s is the *Short Subtree Graph*. We state without proof the following theorem (the proof will be given in the full version of the paper if space permits):

Theorem 1 *The short subtree graph G of an edge-weighted binary tree T is triangulated (that is, it does not contain any simple induced cycles of size greater than three).*

Since the short subtree graph G is triangulated, we can find (in polynomial time, as proven in [5]) a maximal clique separator X that minimizes $max_i|X \cup C_i|$, where $G - X$ is the union of k components $C_1, C_2, \ldots, C_k$. This would allow us to define a decomposition of the dataset into subsets $C_i \cup X$, for $i = 1, 2, \ldots, k$, which would minimize the maximum subset size. Despite the

appeal of such a decomposition (essentially what we did in DCM2), finding such a separator is more time consuming than we want; thus, we used a different technique to find a vertex separator.

DCM3 decomposition First, compute the *short subtree graph*, $G = (V, E)$ on T. Then find a *centroid edge* in T – that is, the edge such that when removed, produces the most balanced bipartition of the leaves. We set X to be the leaves of the short subtree graph around the centroid edge e. (Should this set X fail to be a separator in the short subtree graph defined by the guide tree, we would then resort to computing all maximal clique separators in G; however, in our experience we have never needed to do this.) The subproblems are then defined to be $A_i = X \cup C_i$, where $G - X$ has k distinct connected components, $C_1, C_2, \ldots, C_k$.

Subtrees are then constructed for each subset, A_i, using the "base method", and then combined using the Strict Consensus Merger (see [6, 7]) to produce a tree on the combined dataset. The proof that the resultant tree is accurate (i.e., agrees with the unknown underlying "true tree") follows from the following structural theorem (we omit the proof which is along the same lines as in [6]).

Theorem 2 *Let T be the true tree, let $A_1 \ldots A_k$ be the subproblems obtained in some DCM3 decomposition, perhaps based upon another tree. Suppose that every short quartet in T is a four-clique in some A_i, and that when we apply the base method to each subproblem A_i we obtain the true subtree (i.e. we get $T_i = T|A_i$). Then the Strict Consensus Merger (SCM), a consensus-based supertree method [6], is applied to the set of trees $T_1, T_2, \ldots, T_k$ to yield T.*

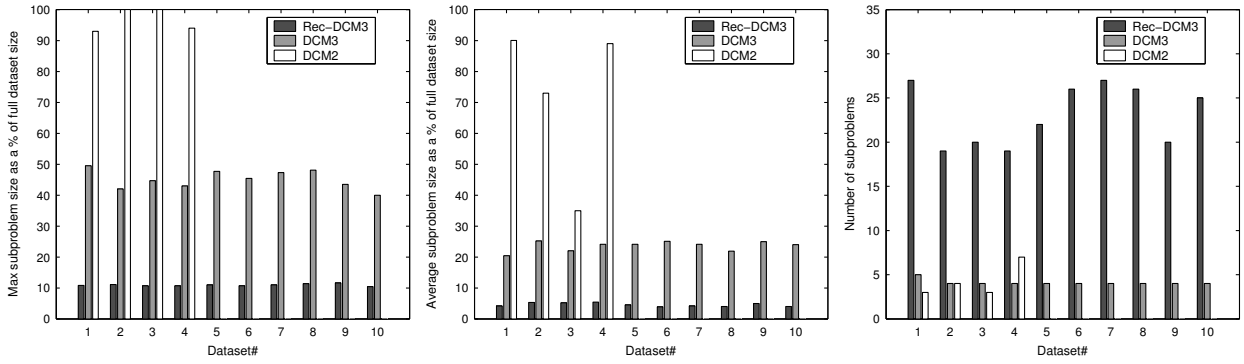
Recursive-DCM3 (Rec-DCM3) decomposition The recursive DCM3 decomposition takes the maximum subproblem size, m, as a parameter and is computed as follows. Let $A_i, \ldots, A_k$ be the computed DCM3 subproblems on S and T. For each subproblem A_i, apply the DCM3 decomposition recursively to A_i with the guide tree defined for subset A_i defined to be T restricted to A_i, until the subproblem becomes at most of size m.

Comparison of decompositions obtained by DCM3, Rec-DCM3, and DCM2: The design of DCM3 was done so as to avoid producing very large subsets. Indeed we see in Figure 1 that DCM3's decomposition produces subproblems of sizes bounded by about half the initial subproblem size, and that Rec-DCM3 produces subproblems that are consistently small (this last observation follows easily from the design of Rec-DCM3, since it always recurses until each subproblem is of size at most one eighth the original size). Note that the DCM2 maximum subproblem sizes are almost equal to the original size of the dataset.

3.2 DCM3 algorithms

All of our DCM3-based algorithms take as input the set $S = \{s_1, \ldots, s_n\}$ of n aligned biomolecular sequences, the base method, and a starting tree T. In these experiments we have used the TNT-Ratchet as our base method, since it is the hardest to improve (PAUP* heuristics are much easier to improve). We describe below the four algorithms which we study.

DCM3 The DCM3 algorithm first computes a DCM3 decomposition on S using the guide-tree T to produce subproblems $A_1, A_2, \ldots, A_k$. It then applies the base method to each subproblem A_i using the guide-tree T restricted to the subset A_i (denoted as $T|A_i$) as the starting tree. The subtrees are then merged using the strict consensus merger (see [7] for description) and the resulting tree is randomly resolved to make it binary, in case it is not.



(a) Maximum subproblem size *(b) Average subproblem size* *(c) Number of subproblems*

Figure 1: Comparison of DCM2, DCM3 and Recursive-DCM3 subproblem statistics. We were unable to compute DCM2 decompositions on datasets 5-10 because the current implementation is slow and uses up too much memory. Thus, no values are reported for DCM2 on these datasets.

Recursive-DCM3 (Rec-DCM3) The Rec-DCM3 algorithm also takes as input the maximum subproblem size m. It operates the same way as DCM3 except that it produces smaller subproblems by recursively applying the DCM3 decomposition until each subproblem is of size at most m. The subtrees are then computed, merged, and randomly resolved (from the bottom-up) to obtain a tree on the full dataset.

Iterative-DCM3 (I-DCM3) I-DCM3 computes a DCM3 tree and follows it up with TBR local search until a local optimum is reached. A DCM3 tree is then computed again using the local optimum as the guide-tree, and this process repeated iteratively for a specified number of iterations, or until a time-limit has expired.

Recursive-I-DCM3 (Rec-I-DCM3) Rec-I-DCM3 is similar to I-DCM3 except that a Rec-DCM3 tree is constructed between iterations instead of a DCM3 tree.

4 Experimental design

Overview Having described our algorithms, we now want to determine whether we can improve upon the TNT-ratchet, possibly the best current technique for solving MP. We focus on performance in the initial 24 hours, and ask the following questions: (1) Which of the DCMs can get closest to "optimal" (i.e., to the current best known MP score) the fastest? (2) Which of the DCMs consistently is able to boost the performance of the parsimony ratchet in TNT? (3) How much of an improvement is gained over the unboosted TNT ratchet, if any? and (4) How long does the best TNT ratchet trial (out of five) take to attain the average MP score obtained at 24 hours by our best DCM? To answer these questions we gathered a number of large biomolecular sequence datasets ranging from a dataset containing slightly more than 1300 sequences, to one containing almost 14,000 sequences. All methods were tested on the basis of five independent runs, on the same platform. Variances were computed to ensure that the results are statistically significant.

Datasets We gathered ten large datasets of the following sizes and types: (1) 1322 lsu rRNA of all organisms (1078 sites) [18], (2) 2000 Eukaryotes rRNA (1326 sites) from the Gutell Lab at

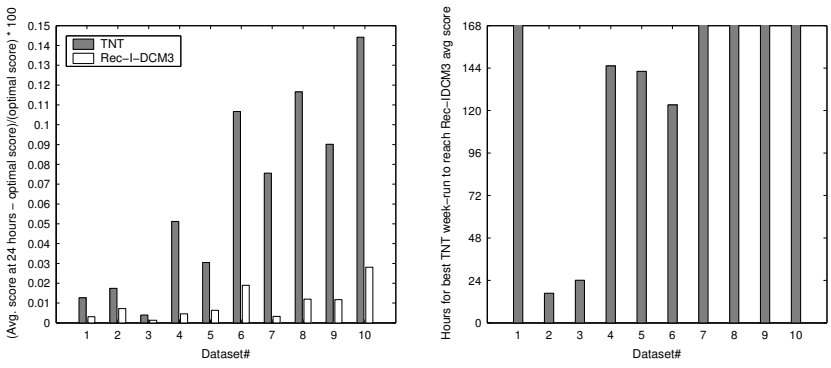
The University of Texas (UT) at Austin, (3) 2594 *rbcL* DNA (1428 sites) [8], (4) 4583 16s rRNA of all Actinobacteria (1263 sites) [1], (5) 6590 ssu rRNA of all Eukaryotes (1661 sites) [18], (6) 7180 three-domain rRNA (1122 sites) from the Gutell Lab at UT Austin, (7) 7233 16s rRNA of all Firmicutes bacteria (1352 sites) [1], (8) 8506 three-domain + two organelles rRNA (851 sites) from the Gutell Lab at UT Austin, (9) 11361 ssu rRNA of all Bacteria (1360 sites) [18], (10) 13921 16s rRNA of all Proteobacteria (1359 sites) [1].

Methods studied For each of the techniques studied here, we used five TNT-ratchet iterations as the base method for the DCMs, and a fast technique to obtain starting trees for all the methods. For the Recursive-I-DCM3 we used subproblems of size at most one-eighth the size of the complete dataset. We studied the following six methods: (1) TNT-ratchet, (2) DCM2(TNT-ratchet), which is a TNT-ratchet search that uses the DCM2 tree as the starting tree, (3) DCM3(TNT-ratchet), which is a TNT-ratchet search that uses the DCM3 tree as the starting tree, (4) Rec-DCM3(TNT-ratchet), which is a TNT-ratchet search that uses the Recursive-DCM3 tree as a starting tree, (5) I-DCM3(TNT-ratchet), and (6) Rec-I-DCM3(TNT-ratchet).

Implementation and Platform Our DCM implementations are a combination of C++ (which uses LEDA 4.3) and Perl scripts. The TNT linux executable was obtained from Pablo Goloboff, one of the authors of TNT. We ran our experiments on three sets of processors running Linux: the phylofarm cluster of 9 dual 500MHz Pentium III processors, 16 dual 733MHz Pentium III processors which are part of the 132-processor SCOUT cluster, and the Phylocluster which consists of 24 dual 1.5GHz AMD Athlon processors; these machines are all at the University of Texas at Austin. For each dataset all the methods were executed on the same cluster, with larger datasets on the faster machines and smaller ones on the slower ones.

5 Results

Overview We set the "optimal" MP score on each dataset to be the best score found over all five runs over all methods in the 24 hour period we allowed; on our datasets, this optimal score was always obtained by Rec-I-DCM3(TNT-ratchet). On each dataset and for each method, we computed the average MP score at hourly intervals, and graphed this value as a percentage deviation from optimality. In our experiments, on every dataset and at every point in time (within these 24 hours), the



(a) *(b)*

Figure 2: (a) compares the average MP score error rates (as a percent above "optimal") obtained after 24 hours by TNT-ratchet and Rec-I-DCM3(TNT-ratchet). (b) shows the time taken by the best TNT trial extended to run for a week, to reach the average Rec-I-DCM3 MP score at 24 hours (bars which reach 128 hours actually indicate that TNT was unable to reach the average Rec-I-DCM3 score at the end of one week).

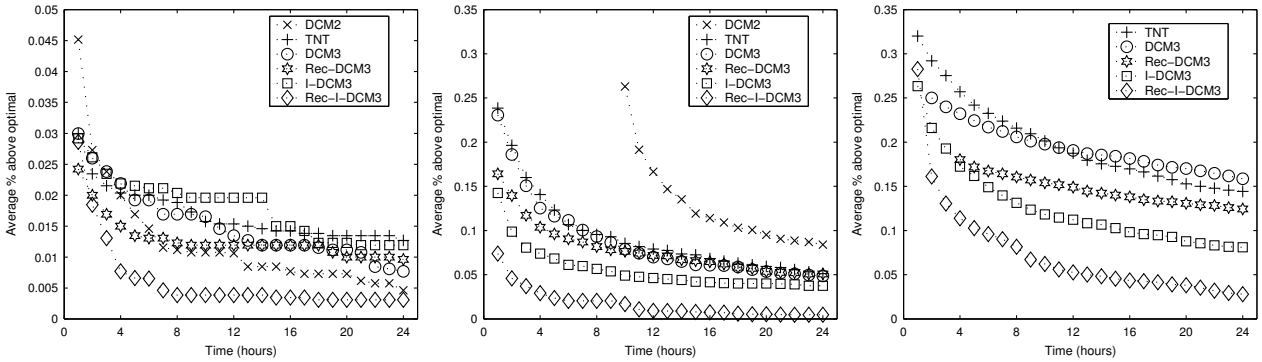
best performance was obtained by Rec-I-DCM3(TNT-ratchet). See Figure 2(a) for a comparison between the performance of Rec-I-DCM3(TNT-ratchet) and TNT-ratchet at 24 hours; this comparison shows that as the dataset size increases, the relative error in MP scores and the gap between the error rates both tend to increase. We then examined how long it would take the best TNT-ratchet trial to reach the average 24 hour score obtained by Rec-I-DCM3(TNT-ratchet), and did an experiment allowing up to one week (168 hours) for that single run. Figure 2(b) shows that on five out of these ten datasets, TNT-ratchet was unable to reach the average Rec-I-DCM3(TNT-ratchet) score in 168 hours, suggesting that order of magnitude improvements are likely on large datasets. The standard deviations of the MP scores at 24 hours for all the methods on all the datasets were very low, at most 0.035% (see the web page for all the standard deviations at 24 hours).

DCM2 vs DCM3 vs unboosted TNT We then compared DCM3 (our new DCM) against DCM2 (our previous DCM), to see whether they were able to boost the performance of the TNT-ratchet (and how they compared relatively). Due to the cost of computing the DCM2 decomposition, we were unable to use DCM2 on datasets 5-10. Figure 3(a) shows that in dataset 1 DCM2 has a slight advantage over DCM3 and the unboosted TNT, but this advantage disappears by dataset 3 (see web page), and it is significantly worse at dataset 4 (where it does not output a tree at all until the 12th hour); this is due to the cost to compute the decomposition (data not shown). DCM2 is therefore unsuitable for these larger datasets.

Comparison of DCM3-based techniques against unboosted TNT We then turned to evaluating the performance of the other variants of DCM3, where we include recursion and/or iteration; see Figure 3. Due to space constraints we only show the results on datasets 1, 4, and 10. On the first dataset (Figure 3(a)) the relative performance of TNT, DCM3, Rec-DCM3, and I-DCM3 varies yet Rec-I-DCM3 stays the consistent winner. Datasets two and three (see web page) follow the same trend. The comparisons on datasets 5-10 are similar to those on datasets 4 and 10 (shown in Figures 3(b) and (c)) and can be viewed on our web page. There we see a consistent improvement in the MP scores of DCMs in the order of DCM3, Rec-DCM3, I-DCM3, and Rec-I-DCM3, with DCM3 being the worse and Rec-I-DCM3 being the best (and significantly improving over the unboosted TNT). Thus, recursion and iteration enhance the performance of DCM3 on these datasets.

6 Summary and Future Plans

In this paper we have presented a new Disk-Covering Method (DCM) for boosting the performance of phylogeny reconstruction methods. Our earlier work (presented in ISMB 1999) presented DCM2 (another DCM), but that technique turned out to have limited ability to boost the performance of the best maximum parsimony heuristics, including TNT's parsimony ratchet. In this study we show how a combination of techniques (including a new dataset decomposition strategy (DCM3), recursion, and iteration) can be used to get very promising results. Our new DCM, which we call Recursive-Iterative-DCM3, dramatically boosts the performance of the default heuristics in the popular software package PAUP*, and quite substantially boosts the performance of TNT's ratchet. We demonstrate our method on 10 real datasets, ranging from 1000 to 14000 sequences, suggesting that we can reduce the time to optimal (as compared to unboosted TNT) by an order of magnitude on many datasets.



(a) Dataset #1 (1322 sequences) *(b) Dataset #4 (4583 sequences)* *(c) Dataset #10 (13921 sequences)*

Figure 3: Average MP scores of all the methods on datasets 1, 4, and 10, given as the percentage above the optimal score. (Graphs on all dataset can be viewed on our web page.) We were unable to use DCM2 on datasets 5-10 because of its computational costs. On datasets 4 and 10 Rec-I-DCM3 is closest to the optimal MP score at every point in time, followed by I-DCM3, Rec-DCM3 and then DCM3 (with the same relative performance on datasets 4-10). Thus, we see that a combination of small subproblems and iteration (Rec-I-DCM3) has the best performance and significantly improves over the unboosted TNT. Note that the range of the y-axes varies across the datasets.

All of the work in this study concerns maximum parsimony, but our results are equally applicable to maximum likelihood. Thus, a study of Rec-I-DCM3 (ML) remains to be conducted — our initial study shows tremendous speed-ups with PAUP* ML heuristics, but better ML heuristics may be harder to boost.

In a related paper (also submitted to this conference) we show that it suffices (in terms of topological accuracy) to get within a hundredth of a percent of the optimal as opposed to finding the exact optimal. The study presented here shows that indeed our best DCM, Rec-I-DCM3, gets within a hundredth of a percent of optimal much faster than the unboosted TNT-ratchet.

7 Acknowledgments

This work was supported by the National Science Foundation under grants DEB 01-20709 (Moret and Warnow), EIA 99-85991 (Warnow, the SCOUT Cluster), EIA 01-13095 (Moret), EIA 01-13654 (Warnow), EIA 01-21377 (Moret), EIA 01-21680 (Warnow), EF 01-31453 (Warnow), EF 01-31654 (Moret), by the David and Lucile Packard Foundation (Warnow), by the Institute for Cellular and Molecular Biology at UT-Austin (Warnow), by the Radcliffe Institute for Advanced Study (Warnow), by the Program in Evolutionary Dynamics at Harvard University (Warnow), and by an Alfred P. Sloan Foundation Postdoctoral Fellowship in Computational Molecular Biology, U.S. Department of Energy DE-FG03-02ER63426 (Williams). We thank Pablo Goloboff for providing us with a copy of the linux version of TNT.

References

[1] B. Maidak et al. The RDP (ribosomal database project) continues. *Nucleic Acids Research*, 28:173–174, 2000.

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

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

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

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

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

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

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

[9] D. R. Maddison. The discovery and importance of multiple islands of most parsimonous trees. *Systematic Biology*, 42(2):200–210, 1991.

[10] 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.

[11] K. C. Nixon. The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics*, 15:407–414, 1999.

[12] D. L. J. Quicke, J. Taylor, and A. Purvis. Changing the landscape: A new strategy for estimating large phylogenies. *Systematic Biology*, 50(1):60–66, 2001.

[13] 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.).

[14] M.J. Sanderson, B.G. Baldwin, G. Bharathan, C.S. Campbell, D. Ferguson, J.M. Porter, C. Von Dohlen, M.F. Wojciechowski, and M.J. Donoghue. The growth of phylogenetic information and the need for a phylogenetic database. *Systematic Biology*, 42:562–568, 1993.

[15] D. E. Soltis, 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. Angiosperm phylogeny inferred from 18s rDNA, *rbcL*, and *atpB* sequences. *Botanical Journal of the Linnean Society*, 133:381–461, 2000.

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

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

[18] J. Wuyts, Y. Van de Peer, T. Winkelmans, and R. De Wachter. The European database on small subunit ribosomal RNA. *Nucleic Acids Research*, 30:183–185, 2002.