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

Usman W. Roshan
Department of Computer Science
University of Texas at Austin
usman@cs.utexas.edu

Bernard M. E. Moret
Department of Computer Science
University of New Mexico
moret@cs.unm.edu

Tandy Warnow
Department of Computer Science
University of Texas at Austin
tandy@cs.utexas.edu
Radcliffe Institute for Advanced Study
twarnow@radcliffe.edu

Tiffani L. Williams
Department of Computer Science
University of New Mexico
tlw@cs.unm.edu

Abstract

Phylogenetic trees are commonly reconstructed based on hard optimization problems such as maximum parsimony (MP) and maximum likelihood (ML). Conventional MP heuristics for producing phylogenetic trees produce good solutions within reasonable time on small datasets (up to a few thousand sequences), while ML heuristics are limited to smaller datasets (up to a few hundred sequences). However, since MP (and presumably ML) is NP-hard, such approaches do not scale when applied to large datasets. In this paper, we present a new technique called Recursive-Iterative-DCM3 (Rec-I-DCM3)*, which belongs to our family of Disk-Covering Methods (DCMs). We tested this new technique on ten large biological datasets ranging from 1,322 to 13,921 sequences and obtained dramatic speedups as well as significant improvements in accuracy (better than 99.99%) in comparison to existing approaches. Thus, high-quality reconstructions can be obtained for datasets at least ten times larger than was previously possible.*

1. Introduction

One of the outstanding problems facing biology is the reconstruction of the "Tree of Life," the evolutionary history of all organisms on this planet. Fundamental to this reconstruction is the ability to produce, within reasonable time constraints, accurate phylogenies for large datasets (tens to hundreds of thousands of taxa), since the "Tree of Life" itself is estimated to contain tens to hundreds of millions of taxa. The most commonly used approaches to phylogeny reconstruction are heuristics for two hard optimization problems, maximum parsimony (MP) and maximum likelihood (ML). However, despite decades of research and algorithm development, acceptably accurate analyses that run within a few days of computation on one processor are not currently possible much beyond a few thousand taxa for MP and a few hundred taxa for ML—nor is it clear that increasing the computing power will enable the analysis of larger datasets, as the accuracy of the heuristics steadily decreases with increasing size of datasets. Polynomial-time algorithms do exist (Neighbor-Joining [19] and UPGMA [12] are the best known examples), but many experimental studies have shown that such trees are not as accurate as those produced by MP or ML analyses.

One of the major challenges confronting both MP and ML heuristics is that the tree space is enormous (there are (2n-5)!! possible solutions for a dataset with n leaves), and existing techniques for searching for optimal trees under both criteria are insufficient for this kind of space. In addition, ML heuristics have to confront the challenge of how to score individual trees - a problem which is easy for MP, but hard for ML. Our research in this paper is focused on addressing the first problem - how to search through tree space; improvements we obtain here will be helpful for both MP and ML analyses.

Whereas 90–95% accuracy is often considered excellent in heuristics for hard optimization problems, heuristics used in phylogenetic reconstruction must be much more accurate: in another study [24], we found that solutions to MP that had an error rate larger than 0.01% (i.e., whose length

Proceedings of the 2004 IEEE Computational Systems Bioinformatics Conference (CSB 2004)
0-7695-2194-0/04 $20.00 © 2004 IEEE

IEEE COMPUTER SOCIETY

exceeded the optimal length by more than 0.01%) produced topologically poor estimates of the true tree. Thus, heuristics for MP need at least 99.99% accuracy (and probably significantly more on very large datasets) in order to produce topologically accurate trees. Obtaining this level of accuracy while running within a reasonable time presents a stiff challenge to algorithm developers.

In this paper, we describe a new technique that makes it possible to reach that level of accuracy on datasets of large size—indeed, of sizes at least one order of magnitude larger than could be analyzed before. Our technique is called *Recursive-Iterative DCM3 (Rec-I-DCM3)*, a divide-and-conquer technique that combines recursion and iteration with a new variant of the *Disk-Covering Method (DCM)* to find highly accurate trees quickly. *Rec-I-DCM3* uses iteration for escaping local optima, the divide-and-conquer approach of the DCMs to reduce problem size, and recursion (as pioneered in [22]) to enable further localization and reduction in problem size. A *Rec-I-DCM3* search not only dramatically reduces the size of the explored tree space, but also finds a larger fraction of MP trees with better scores than other methods. We demonstrate the power of *Rec-I-DCM3* on ten large biomolecular sequence datasets, each containing more than 1,000 sequences (half contain over 6,000 sequences and the largest contains almost 14,000 sequences). Our study shows that *Rec-I-DCM3* convincingly outperforms TNT [5]—the best implemented MP heuristic—often by orders of magnitude, on all datasets and at all times during the time period (usually 24 hours) allotted for computation. Although we focused on testing our technique on MP analyses, our improvements work as well in ML analyses.

2. Our Datasets

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

1. A set of 1,322 aligned large subunit ribosomal RNA of all organisms (1,078 sites) [25].
2. A set of 2,000 aligned Eukaryotes ribosomal RNA sequences (1,326 sites) obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.
3. A set of 2,594 *rbcL* DNA sequences (1,428 sites) [9].
4. A set of 4,583 aligned 16s ribosomal Actinobacteria RNA sequences (1,263 sites) [11].
5. A set of 6,590 aligned small subunit ribosomal Eukaryotes RNA sequences (1,661 sites) [25].
6. A set of 7,180 aligned ribosomal RNA sequences (1,122 sites) from three phylogenetic domains obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.
7. A set of 7,233 aligned 16s ribosomal Firmicutes (bacteria) RNA sequences (1,352 sites) [11].
8. A set of 8,506 aligned ribosomal RNA sequences (851 sites) from three phylogenetic domains, plus organelles (mitochondria and chloroplast), obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.
9. A set of 11,361 aligned small subunit ribosomal Bacteria RNA sequences (1,360 sites) [25].
10. A set of 13,921 aligned 16s ribosomal Proteobacteria RNA sequences (1,359 sites) [11].

3. Maximum Parsimony

Let S be a set of n sequences, each of length n, over a fixed alphabet Σ. Let T be a tree leaf-labelled by the set S and with internal nodes labelled by sequences of length n over Σ. The *length* (or *parsimony score*) 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 equal length is just the number of positions in which the two strings differ.) Thus the length of a tree is also the total number of point mutations along the edges of the tree. The *Maximum Parsimony (MP)* problem seeks the tree T leaf-labelled by S with the minimum length. While MP is NP-hard [4], constructing the optimal labeling of the internal nodes of a fixed tree T can be done in polynomial time [3].

3.1. Iterative Improvement Methods

Iterative improvement methods are some of the most popular heuristics in phylogeny reconstruction. A fast technique is used to find an initial tree, then a local search mechanism is applied repeatedly in order to find trees with a better score. The most commonly used local move is called *Tree-Bisection and Reconnection (TBR)* [10]. In TBR, an edge is removed from the given tree T, thereby creating two subtrees, t and $T - t$; the two subtrees are then reconnected by subdividing two edges (one in each subtree) and adding an edge between the newly introduced nodes.

Heuristics for MP are implemented in many software packages, but the most popular package is PAUP*[21]. The software package TNT, however, provides a faster implementation of local search and also implements search

IEEE COMPUTER SOCIETY

strategies such as simulated annealing and genetic algorithms. The default (recommended) technique is a combination of simulated annealing, divide-and-conquer, and genetic algorithmic techniques. In our unpublished studies the TNT-default technique is faster and more accurate than the PAUP*-default technique for solving MP; therefore we focus on TNT in this study. Both, PAUP* and TNT, are publicly available but neither is open-source.

3.2. Disk-Covering Methods

Disk-Covering Methods (DCMs) [7, 8, 15, 18, 23] are a family of divide-and-conquer methods designed to "boost" the performance of existing phylogenetic reconstruction methods. All DCMs proceed in four major phases: (i) decomposing the dataset, (ii) solving the subproblems, (iii) merging the subproblems, and (iv) refining the resulting tree. Variants of DCMs come from different decomposition methods—the last three phases are unaffected. The first DCM [7], 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 [23]. The second DCM [8], also called DCM2, was designed to speed up heuristic searches for MP trees; we showed that when DCM2 was used with PAUP*-TBR search, it produced better trees faster on simulated datasets.

4. DCM3

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

4.1. The optimal decomposition

We begin by describing the algorithm to find an *optimal* DCM3 decomposition. (As noted, it is not the decomposition used in our implementation, but we need it to establish the framework.) Consider a tree T on our set S of taxa and an edge weighting w of T, w: $E(T) \rightarrow \Re^+$. (A possible edge weighting is given by the Hamming distances under the MP labelling of the nodes of T.) We construct the *short subtree graph*, which is the union of cliques formed on "short subtrees" around each edge. Let e be an internal edge (not touching a leaf) in T; then removing e and its two endpoints from T breaks T into four subtrees. A *short quartet* around e is composed of four leaves, one from each of these four subtrees, where each leaf is selected to be the closest (according to the edge weights) in its tree to e. This short quartet need not be unique: several leaves in the same subtree may lie at the same shortest distance from e. Thus we define the *short subtree* around e to be the set $X(e)$ of all leaves that are part of a short quartet around e. We will use the *clique* on $X(e)$: the graph with $X(e)$ as its vertices and with every pairwise edge present, weighted according to w; denote this clique by $K(e)$. The *short subtree graph* is then the union, over all internal edges e of the guide tree, of the $K(e)$. We now use the following lemma from [2] and [6] to prove that the short subtree graph is triangulated.

Lemma 1. *Every triangulated graph is the intersection graph of subtrees of a tree and vice-versa.*

Theorem 1. *The short subtree graph G of an edge-weighted binary tree T is triangulated.*

Proof. We will use Lemma 1 to show that $G = (V, E)$ is isomorphic to such an intersection graph. First, we define a subtree for each $v \in V$. Create a copy of T called T' and subdivide every internal edge of T'; thus, we create a new node v_e for every internal edge e.

Recall that each node $v \in V$ represents a leaf in $L(T)$, where T is the guide-tree. For each node $v \in V$, consider the set of edges $E_v(T)$ such that for all $e \in E_v(T)$, v is a leaf in the short subtree of e. Note that $E_v(T)$ is the set of edges in the original guide-tree T. We now define for each $e \in E_v(T)$ the path $P(v, v_e)$ in T' which begins at leaf v and ends at v_e. Clearly, this path is a subtree of T'. Let $P(v)$ be the union of all such paths in T' that begin at v and end at v_e, for all edges $e \in E_v(T)$. Note that $P(v)$ is a subtree of T' which we associate to v in the short subtree graph; we call this subtree t_v.

IEEE COMPUTER SOCIETY

We now need to show that $(u, v) \in E$ iff $t_u \cap t_v \neq \emptyset$. By definition of G, this is equivalent to showing that u, v belong to some same short subtree iff $t_u \cap t_v \neq \emptyset$. So assume that u and v belong to some common short subtree, defined by some edge e. By definition, the subtrees t_u and t_v will share at least the node v_e; thus, $t_u \cap t_v \neq \emptyset$. On the other hand, suppose that t_u and t_v share some vertex x. There are two cases to consider, depending on whether $x = v_e$ for some edge e or not. When $x = v_e$, it is easy to see that both u and v are in a short subtree around edge e; hence $(u, v) \in E$. On the other hand, suppose $x \neq v_e$ for any edge e. Then x is a node in T but not in T'. By construction, t_u and t_v are each the union of paths that end in the newly introduced nodes; hence, t_u has a node $v_{e'}$ adjacent to x, and t_v has a node $v_{e''}$ also adjacent to x. Note that if $e' \neq e''$, then $v_{e''}$ is on the path from u to x, and hence $v_{e''}$ is in t_u. But then $v_{e''} \in t_u \cap t_v$. □

Since the short subtree graph G is triangulated, we can find in polynomial time, as proved in [6], a maximal clique separator X that minimizes $max_i|X \cup C_i|$, where $G - X$ is the union of m components $C_1, C_2, \ldots, C_m$. Using X, we can define the *Optimal DCM3 Decomposition* to be formed of the subsets $C_i \cup X$, for $i = 1, 2, \ldots, m$. Its computation thus proceeds in two steps:

1. Construct the short subtree graph.
2. For each of the maximal cliques in the short subtree graph, determine if it is a separator. If so, then compute the maximum size of any created subproblem and choose that separator which minimizes it.

Theorem 2. *The Optimal DCM3 decomposition can be computed in $O(n^3)$ time.*

Proof. Constructing the short subtree graph takes $O(n^3)$ time. There are $O(n)$ maximal cliques [6]. For each maximal clique, determining whether it is a separator and, if so, computing the sizes of the components, takes a total of $O(n^3)$ time. □

Constructing the short subtree graph and finding the separator and connected components are the costliest parts of this decomposition. Although both parts have $O(n^3)$ running times, finding the optimal separator and the connected components takes much longer in practice. Thus, the optimal decomposition is too expensive to compute for the kind of n we have in mind—even if we can obtain one optimal decomposition relatively fast for one tree on a million taxa, we will need to iterate this computation many, many times.

4.2. A fast suboptimal decomposition

Rather than explicitly seeking a clique separator X in G which minimizes the size of the largest subproblem, we apply a simple heuristic to get a decomposition, which in practice turns out to be a good decomposition. Although finding such a clique separator in a triangulated graph takes polynomial time, it is still costly in practice and is the reason why DCM2's decomposition was expensive to compute.

We first find a *centroid edge* in T—that is, an edge such that when removed, produces the most balanced bipartition of the leaves. This can be found in linear time using a depth-first search on T. We set X to be the short subtree around the centroid edge e. Under most conditions (other than the case where the guide tree is ultrametric, i.e., when the strong molecular clock hypothesis holds), the short subtree will be a separator and it tends to produce subproblems that are acceptably small—in all our runs the short subtree was a separator. 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, we never needed to do this.

Once we have computed the short subtree graph, it takes $O(n^2)$ time to find the subproblems using the centroid technique—we just perform a depth first search on the short subtree graph to find the connected components. However, computing the short subtree can take $O(n^3)$ time in the worst case, i.e., when the short subtrees are of size $O(n)$. In our experiments, however, the short subtrees are usually of size 4 or 5; thus, reducing the running time to $O(n^2)$.

4.3. Comparison of DCM decompositions

We designed DCM3 in part to avoid producing large subsets, as DCM2 is prone to do. Yet, of course, the subproblems produced from a very large dataset remain too large for immediate solution by a base method. Hence we used the approach successfully pioneered by Tang and Moret with DCM-GRAPPA [22] and used DCM3 recursively, producing smaller and smaller subproblems until every subproblem was small enough to be solved directly. Figure 1 shows that DCM3 produces subproblems of sizes bounded by about half the initial subproblem size and much smaller than those produced by DCM2. (Rec-DCM3 in this series of tests was set up to recurse until each subproblem was of size at most one quarter of the original size.)

4.4. Subtree construction and assembly

Once the dataset is decomposed into overlapping subsets $A_1, A_2, \ldots, A_m$ (for us, $m \leq 4$ is typical), subtrees are constructed for each subset, A_i, using the chosen "base method," and then combined using the Strict Consensus Merger [7, 8] to produce a tree on the combined dataset. The proof that the resulting tree is accurate (i.e., agrees, with high probability and in the limit, with the unknown underlying "true tree") follows from the following structural the-

IEEE COMPUTER SOCIETY

orem (we omit the proof which is along the same lines as in [7]).

Theorem 3. *Let T be the true tree and let $A_1, A_2, \ldots, A_m$ be the subproblems obtained in some DCM3 decomposition. If every short quartet in T is a four-clique in some A_i and if the base method applied to A_i returns the true tree for that subset (i.e., $T_i = T|A_i$), then the strict consensus merger of trees $T_1, T_2, \ldots, T_k$ yields the true tree T.*

4.5. Rec-I-DCM3

Our *Rec-I-DCM3* algorithm takes as input the set $S = \{s_1, \ldots, s_n\}$ of n aligned biomolecular sequences, the chosen base method, and a starting tree T. In our experiments, we have used TNT (with default settings) as our base method, since it is the hardest to improve (in comparison, the PAUP* implementation of the parsimony ratchet [1] is easier to improve). Our algorithm produces smaller subproblems by recursively applying the centroid-edge decomposition until each subproblem is of size at most k; in our experiments we used subproblems of size at most one quarter of the original size. The subtrees are then computed, merged, and resolved (from the bottom-up, using random resolution) to obtain a binary tree on the full dataset. These steps are repeated for a specified number of iterations.

5. Experimental design

Having designed a new algorithm and having preliminary evidence (on small simulations) that it outperforms existing techniques, we now need to evaluate its performance in practice. The experimental evaluation of algorithms for phylogenetic reconstruction is a difficult endeavor (see [13, 14] for details). Because credible simulations of evolution remain lacking at the scale of 10,000 or more taxa, we chose to use biological datasets in our study. This choice ensures biological relevance of our results, but it prevents us from evaluating the accuracy of reconstructed trees, since the "true" tree is not available. However, other work from our group [24] tells us that we need to achieve excellent approximation of the parsimony score (tree length) in order to have any chance at reconstructing the true topology. Thus, we focused our testing on the quality of approximation in terms of the parsimony score.

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

Implementation and platform: Our DCM implementations are a combination of LEDA, C++, and Perl scripts. The TNT Linux executable was obtained from Pablo Goloboff, one of the authors of TNT. We ran our experiments on three sets of processors, all running Linux: the `Phylofarm` cluster of 9 dual 500MHz Pentium III processors; a part of the 132-processor `SCOUT` cluster, consisting of 16 dual 733MHz Pentium III processors, and the `Phylocluster` of 24 dual 1.5GHz AMD Athlon processors, all at the University of Texas at Austin. For each dataset all the methods were executed on the same cluster; larger datasets were run on the faster machines. Note that Rec-I-DCM3(TNT) was not parallelized: all subproblems were processed one after the other even though we had access to many processors at once. Clearly we can expect even greater speedups if we processed all the subproblems in parallel.

6. Results

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

Figure 2 shows the performance of *Rec-I-DCM3(TNT)* and of TNT at 24 hours. As the size of the dataset increases, the relative error in MP scores increases, but at a much faster rate for TNT than for *Rec-I-DCM3(TNT)*, so that the accuracy gap between the two increases quite rapidly.

Figure 2(a) indicates how long it took TNT, in the best of five runs, to match the average scores obtained by *Rec-I-DCM3(TNT)* after 24 hours—we stopped the clock after one week of computation if the TNT run had not achieved a match by then, something that happened on the seven largest datasets. On these seven datasets, we plotted the ratio of the time taken by Rec-I-DCM3(TNT) to reach the best TNT week-run score against the time taken by the TNT week-run to reach that score. Figure 3(b) shows that Rec-I-DCM3(TNT) reaches the best TNT week-run score at least

IEEE COMPUTER SOCIETY

10 times faster on Datasets 4-10. On Datasets 6 and 10 the improvement is *50* times.

Figure 4 compares the time-dependent behaviors of TNT and *Rec-I-DCM3(TNT)* on our three smallest datasets (1, 2, and 3), while Figure 5 shows the same for the three medium datasets (4, 5, and 6), and Figure 6 shows the same for our three large datasets (8, 9 and 10). Figure 7 shows the performance of TNT and *Rec-I-DCM3(TNT)* on our largest dataset of about 14,000 sequences. It should be noted that the 24-hour time limit was perhaps overly limiting for the largest dataset: a quick look at the curves appears to indicate that even *Rec-I-DCM3(TNT)* has not yet reached a plateau at that point. The improvement achieved by boosting TNT with *Rec-I-DCM3* is significant on all datasets as well as at all time intervals. In particular, note that the boosted version of TNT shows much stronger decreases in MP scores in the first several hours than the unboosted version. The low variances, as indicated by the datapoints of all five runs of both the methods, show that our results are statistically sound. Note that there is a noticeably large difference between the worse datapoint of Rec-I-DCM3(TNT) and the best datapoint of TNT at 24 hours on Dataset 6 onwards.

Figure 8 shows how the error rate (deviation above the best score) of *Rec-I-DCM3(TNT)* decreases with computation time on each of the ten datasets. While the initial trees computed for the large datasets tend to exhibit large error (as large as 0.35%), the error drops very rapidly—even more rapidly for the large datasets than for the smaller ones. Thus, not only do the error rates of *Rec-I-DCM3(TNT)* fall more rapidly than those of TNT alone, but they have a positive second derivative: the larger they are, the faster they fall.

7. Related Work and Discussion

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 best known score on various real datasets, preferably of at least one hundred sequences. These studies [5, 16, 17, 18, 20] have suggested that the "parsimony ratchet" [16], an iterated local search technique, was more effective than Tree-Bisection and Reconnection (TBR) hill-climbing [10], and that TNT (which uses the parsimony ratchet) was more efficient than PAUP*'s implementation of the ratchet. The study by Pablo Goloboff [5] suggested that the TNT-default technique (a combination of simulated annealing, divide-and-conquer, and genetic algorithms) was better than the TNT-ratchet implementation, especially on large datasets.

Our previous studies of our DCM variants included a study of the performance of boosted PAUP* MP heuristics using DCM2, which showed good results [8]. However, our own analyses of phylogenetic reconstruction methods showed convincingly that the default PAUP* heuristic was not as powerful as the default TNT heuristics; they also showed that DCM2 failed to boost TNT. DCM3 was our attempt to remedy this situation, but it, too, failed to boost TNT reliably. The two key ideas were to use (i) iteration in order to refine the guide tree (since, obviously, the initial guide tree is perforce poor), and (ii) recursion, which had been used with spectacular success by Tang and Moret [22] in boosting the GRAPPA reconstruction software (for gene-order data) over three orders of magnitude.

Looking over our experience with both *Rec-I-DCM3* and DCM-GRAPPA, we can make reasonable inferences as to the reasons for the success of both. Recursion is necessary to bridge the gap between datasets that can be analyzed today by the base method (2,000–3,000 sequences for TNT, 14–15 genomes for GRAPPA) and large datasets. DCM-GRAPPA bridged three orders of magnitude in simulation studies (from 14 to 1,200 taxa); our *Rec-I-DCM3* bridges a least one order, but may be capable of more—biological datasets with 100,000 aligned sequences do not yet exist! But speed is only one component of the equation: the other is accuracy. In the case of DCM-GRAPPA, which uses no iteration and a simple DCM1 decomposition, the accuracy derives in good part from the excellent properties of gene-order data in phylogenetic reconstruction—and only simulations were used. In our case, recursion alone (a version we could call *Rec-DCM3*, which we tested), does not suffice: we need the ability, through iteration, to modify the guide tree at every level of the recursion. The combination of the two has synergistic effects: the decomposition is dictated by the guide tree, and thus by the successive iterations, and localizes the work for the recursive stages and the base method; and recursion enables the same process to take place at finer and finer levels of resolution, while communicating the results back to the coarser levels. Our results suggest that DCM-GRAPPA could be significantly enhanced by using a version of *Rec-I-DCM3*, but also that a more discriminating use of the guide tree, yielding more than just four subsets (e.g., by using a type of DCM1 decomposition guided by the guide tree), would probably work better for us, especially when tackling truly large datasets (on the order of millions of sequences) where several orders of magnitude will have to be bridged.

8. Summary and Future Plans

We have presented a new Disk-Covering Method for boosting the performance of phylogenetic reconstruction methods. Our new technique, *Rec-I-DCM3*, dramatically

IEEE COMPUTER SOCIETY

improves on existing techniques for maximum parsimony, as our experimental study on ten large biological datasets shows. *Rec-I-DCM3* reconstructs trees with MP scores that are within 0.01% of the best score—a necessary feature since only MP estimates that are 99.99% correct or better can yield good topological estimates of the true tree. Moreover, *Rec-I-DCM3* produces high-quality reconstructions very quickly (within 24 hours) even on the largest datasets. Other approaches fail to achieve such accuracy within a reasonable time period. While the work in this study focused on maximum parsimony, our results are equally applicable to maximum likelihood (which has far more severe time constraints, so that a much much larger payoff in running time is likely) and to phylogenetic multiple sequence alignment. An open source C implementation of our algorithm is in progress and will be released in the near-future.

9. Acknowledgments

This work was supported by the National Science Foundation under grants ANI 02-03584 (Moret), DEB 01-20709 (Moret and Warnow), EF 03-31453 (Warnow), EF 03-31654 (Moret), EIA 99-85991 (Warnow, the SCOUT Cluster), IIS 01-13095 (Moret), IIS 01-13654 (Warnow), IIS 01-21377 (Moret), IIS 01-21680 (Warnow), by an Alfred P. Sloan Foundation Postdoctoral Fellowship in Computational Molecular Biology DE-FG03-02ER63426 (Williams), 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), and by the Program in Evolutionary Dynamics at Harvard University (Warnow). We thank Pablo Goloboff for providing us with a Linux executable for TNT.

References

[1] O. Bininda-Emonds. Ratchet implementation in PAUP*4.0b10, 2003. Available from www.tierzucht.tum.de:8080/WWW/Homepages/Bininda-Emonds.

[2] P. Buneman. A characterization of rigid circuit graphs. *Discrete Mathematics*, 9:205–212, 1974.

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

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

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

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

[7] 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.

[8] 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.

[9] M. Kallerjo, J. Farris, M. Chase, B. Bremer, and M. 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.

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

[11] B. Maidak, J. Cole, T. Lilburn, C. P. Jr, P. Saxman, J. Stredwick, G. Garrity, B. Li, G. Olsen, S. Pramanik, T. Schmidt, and J. Tiedje. The RDP (ribosomal database project) continues. *Nucleic Acids Research*, 28:173–174, 2000.

[12] C. Michener and R. Sokal. A quantitative approach to a problem in classification. *Evolution*, 11:130–162, 1957.

[13] B. Moret. Towards a discipline of experimental algorithmics. In M. Goldwasser, D. Johnson, and C. McGeoch, editors, *Data Structures, Near Neighbor Searches, and Methodology: Fifth and Sixth DIMACS Implementation Challenges*, volume 59 of *DIMACS Monographs*, pages 197–213. AMS Press, 2002.

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

[15] 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.

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

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

[18] U. Roshan, B. Moret, T. Williams, and T. Warnow. Performance of supertree methods on various dataset decompositions. In O. Bininda-Emonds, editor, *Phylogenetic Supertrees: Combining Information to Reveal the Tree of Life*, volume 3 of *Computational Biology*, pages 301–328. Kluwer Academic Publishers, 2004.

[19] N. Saitou and M. Nei. The neighbor-joining method: A new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[20] D. Soltis, P. Soltis, M. Chase, M. Mort, D. Albach, M. Zanis, V. Savolainen, W. Hahn, S. Hoot, M. Fay, M. Axtell, S. Swensen, L. Prince, W. Kress, K. Nixon, and J. Farris. Angiosperm phylogeny inferred from 18s rDNA, *rbcL*, and *atpB* sequences. *Botanical Journal of the Linnean Society*, 133:381–461, 2000.

IEEE COMPUTER SOCIETY

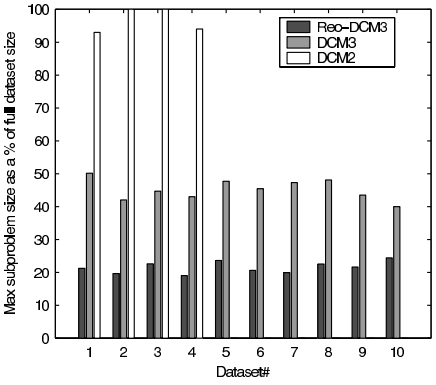
[21] D. Swofford. PAUP*: Phylogenetic analysis using parsimony (and other methods), 2002. Sinauer Associates, Sunderland, Mass., Version 4.0.

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

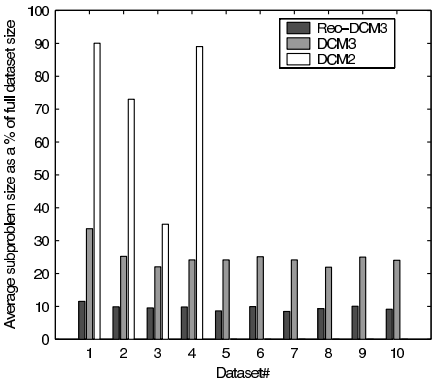
[23] T. Warnow, B. Moret, and K. St. John. Absolute convergence: True trees from short sequences. In *Proc. 12th Ann. ACM-SIAM Symp. Discrete Algorithms (SODA'01)*, pages 186–195. SIAM Press, 2001.

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

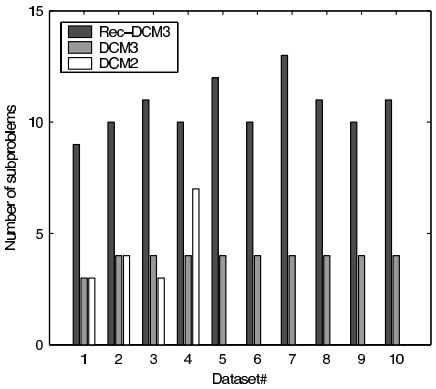
[25] J. Wuyts, Y. V. de Peer, T. Winkelmans, and R. D. Wachter. The European database on small subunit ribosomal RNA. *Nucleic Acids Research*, 30:183–185, 2002.



(a) maximum relative subproblem size



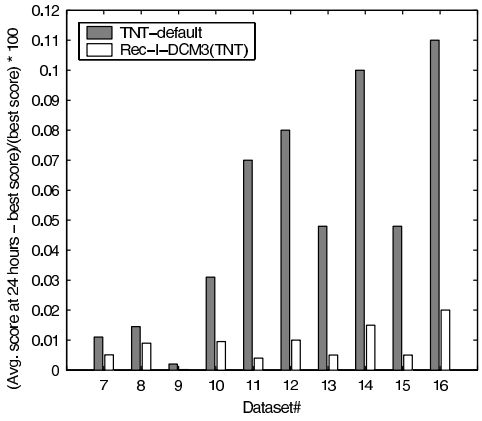
(b) mean relative subproblem size



(c) number of subproblems

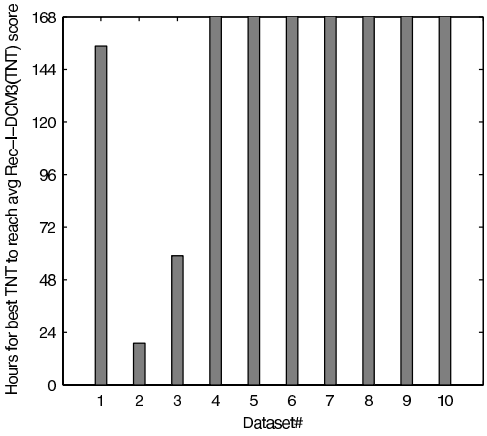
Figure 1. Comparison of DCM2, DCM3 and Recursive-DCM3 decompositions. DCM2 decompositions on Datasets 5–10 could not be computed due to memory limitations.

IEEE COMPUTER SOCIETY

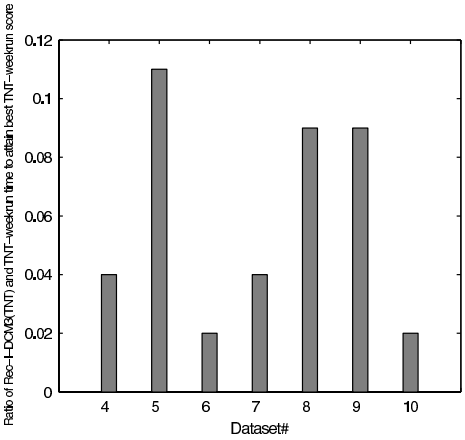


deviation from optimal

Figure 2. Shown here is the average deviation above best score after 24 hours by TNT and *Rec-I-DCM3(TNT)*.



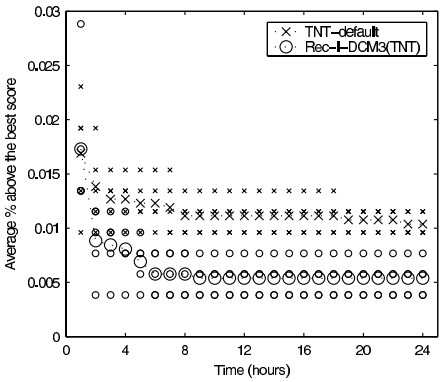
(a) TNT time to best score



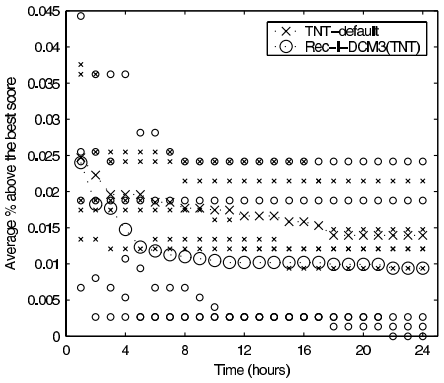
(b) Improvement of Rec-I-DCM3 over TNT week-run

Figure 3. Part (a) shows the time taken by the single best TNT trial, extended to run for up to a week, to match the average *Rec-I-DCM3(TNT)* score at 24 hours—bars that reach the top indicate that TNT could not reach a match after a week of computation. Part (b) shows the ratio of the time taken by Rec-I-DCM3(TNT) to find the best TNT-default week-run score against the time taken by the TNT-default week-run to find that score. We show the improvement only on Datasets 4-10 because that is where the best score found by the TNT-default week-run was worse than the mean Rec-I-DCM3(TNT) score at 24 hours.

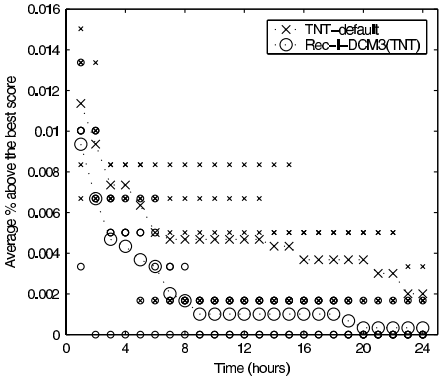
IEEE COMPUTER SOCIETY



(a) Dataset #1 (1,322 sequences)

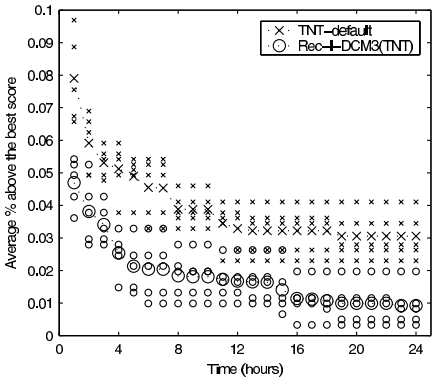


(b) Dataset #2 (2,000 sequences)

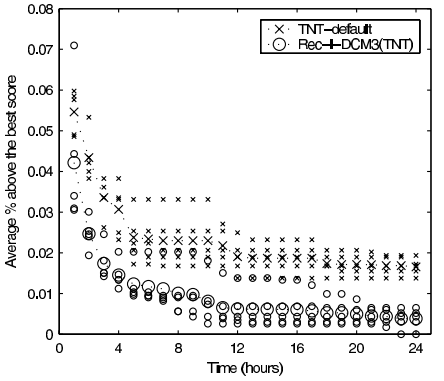


(c) Dataset #3 (2,594 sequences)

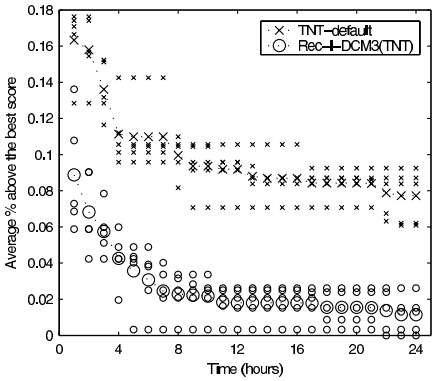
Figure 4. Average MP scores of TNT and *Rec-I-DCM3(TNT)* on Datasets 1, 2, and 3, given as the percentage above the best score. Shown in each graph are datapoints of all five runs of both methods indicated by the small symbols. The variances are low in Datasets 1 and 3 and moderate in Dataset 2. Note: the vertical range varies across the datasets.



(a) Dataset #4 (4,584 sequences)



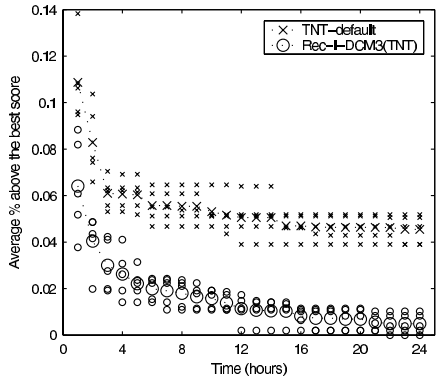
(b) Dataset #5 (6,590 sequences)



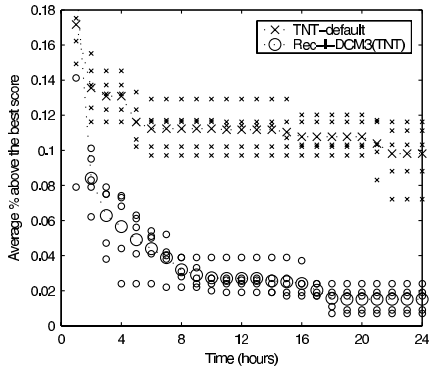
(c) Dataset #6 (7,180 sequences)

Figure 5. Average MP scores of TNT and *Rec-I-DCM3(TNT)* on Datasets 4, 5, and 6, given as the percentage above the best score. Shown in each graph are datapoints of all five runs of both methods indicated by the small symbols. The variances are low on all the datasets shown here with little overlap of points in Datasets 4 and 5 and almost no overlap in Dataset 6. Note: the vertical range varies across the datasets.

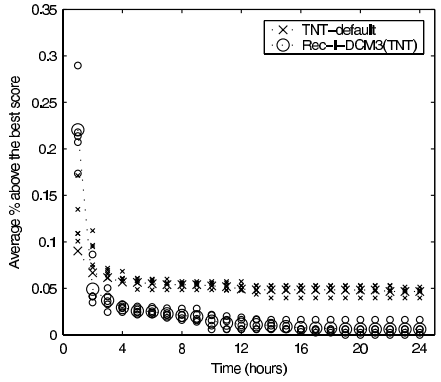
IEEE COMPUTER SOCIETY



(a) Dataset #7 (7,322 sequences)

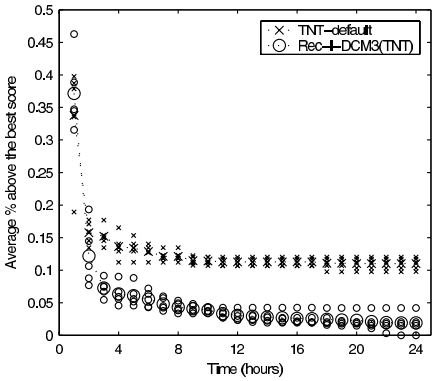


(b) Dataset #8 (8,506 sequences)



(c) Dataset #9 (11,361 sequences)

Figure 6. Average MP scores of TNT and *Rec-I-DCM3(TNT)* on Datasets 8, 9, and 10, given as the percentage above the best score. Shown in each graph are the datapoints of all five runs of both methods indicated by small symbols. After the fourth hour there is no overlap of points and the variances of both the methods are low. Note: the vertical range varies across the datasets.



(a) Dataset #10 (13,921 sequences)

Figure 7. Average MP scores of TNT and *Rec-I-DCM3(TNT)* on Dataset 10, given as the percentage above the best score. Also shown are the datapoints of all five runs of both methods indicated by small symbols. Note that the variances are very low and after the third hour there is no overlap of points.

IEEE COMPUTER SOCIETY

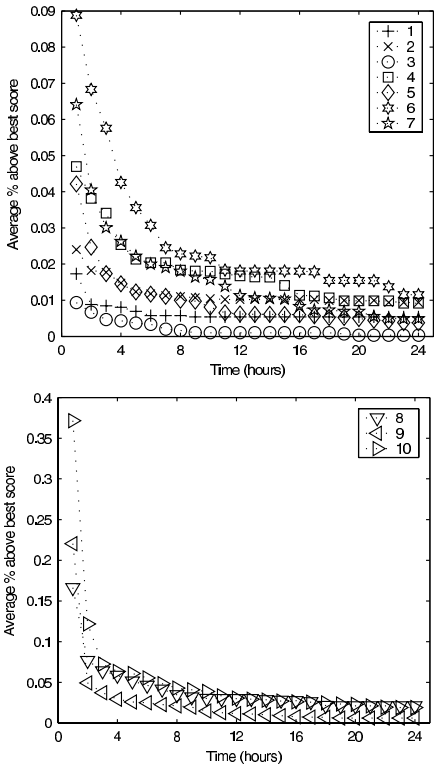


Figure 8. Decrease in error rates with time on all datasets for *Rec-I-DCM3(TNT)*

IEEE COMPUTER SOCIETY