PRec-I-DCM3: A Parallel Framework for Fast and Accurate Large Scale Phylogeny Reconstruction*

Cristian Coarfa† Yuri Dotsenko† John Mellor-Crummey†
Luay Nakhleh† Usman Roshan‡

Abstract

Phylogenetic trees play a major role in representing the evolutionary relationships among groups of organisms. Their accurate reconstruction very often involves solving hard optimization problems, particularly the maximum parsimony (MP) and maximum likelihood (ML) problems. Various heuristics have been devised for solving these two problems; yet, they obtain good results within reasonable time limits only on small datasets. This has been a major limitation for large scale phylogeny reconstruction, and particularly efforts for assembling the Tree of Life—the evolutionary relationship of all organisms on earth. Roshan *et al.* have recently introduced `Rec-I-DCM3`, an efficient and accurate meta-method for solving the MP problem on large datasets of up to 14,000 taxa. Nonetheless, a drastic improvement in `Rec-I-DCM3`'s performance is still needed in order to achieve similar (or better) accuracy on datasets at the scale of the Tree of Life. In this paper we address this issue in two ways. We investigate, through experiments on biological datasets, the optimal choice of parameters whose values affect the performance of `Rec-I-DCM3`. Further, we improve the performance of `Rec-I-DCM3` via parallelization. Experimental results demonstrate that our parallel method, `PRec-I-DCM3`, achieves significant improvements, both in speed and accuracy, over its sequential counterpart.

1 Introduction

Phylogenies play a major role in representing the evolutionary relationships among groups of taxa. Their pervasiveness has led biologists, mathematicians, and computer scientists to develop a wide array of methods for their reconstruction. One of the outstanding problems facing biology today is

*This work was supported in part by the Department of Energy under Grant DE-FC03-01ER25504/A000. The computations were performed on an Itanium cluster purchased with support from the NSF under Grant EIA-0216467, Intel, and Hewlett Packard.

†Department of Computer Science, Rice University, Houston, TX 77005, USA. {ccristi,dotsenko,johnmc,nakhleh}@cs.rice.edu

‡Department of Computer Science, New Jersey Institute of Technology, Newark, NJ 07102, USA. usman@cs.njit.edu

the reconstruction of the Tree of Life—the evolutionary history of all organisms on earth. 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 [18] and UPGMA [11] 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. As a result of the improved accuracy of MP approaches over polynomial-time methods and of the significantly lower cost of MP analyses as compared to ML analyses, which are sometimes more accurate, the majority of published phylogenies to date have been derived using MP-based heuristics [19].

Whereas 90–95% accuracy is often considered excellent in heuristics for hard optimization problems, heuristics used in phylogenetic reconstruction must be much more accurate: Williams and Moret found that solutions to MP that had an error rate larger than 0.01% (i.e., whose length exceeded the optimal length by more than 0.01%) produced topologically poor estimates of the true tree [22]. 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 [17], Roshan *et al.* presented 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. Their technique, called *Recursive-Iterative DCM3* (`Rec-I-DCM3`), employs a divide-and-conquer strategy 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 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. Roshan *et al.* demonstrated 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). Their study showed that `Rec-I-DCM3` convincingly outperformed `TNT` [6]—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.

In order to be able to handle large datasets at the scale of the Tree of Life within reasonable time limits and with high accuracy, the performance of `Rec-I-DCM3` has to be improved by orders of magnitude. In this paper, we address the problem of large scale phylogenetic tree reconstruction by building on the success of `Rec-I-DCM3`. We have investigated, through experiments on biological datasets, the best choice of parameters whose values affect the performance

of Rec-I-DCM3. Further, we have designed and implemented a parallel version of the method, called PRec-I-DCM3. We have tested PRec-I-DCM3 on the biological datasets used in [17]. The results we obtained show a drastic improvement in the performance of the method. For example, on the largest dataset used by Roshan *et al.*, Rec-I-DCM3 took about 13 hours to find the MP tree found by PRec-I-DCM3 within less than three hours. Further, the parsimony scores of trees computed by PRec-I-DCM3 are consistently better than those computed by Rec-I-DCM3 within the same amount of time.

2 Maximum Parsimony

The parsimony criterion is but a reflection of Occam's razor: the tree with the minimum number of mutations along its branches best explains the data. In this section, we review the formal definition of the maximum parsimony problem and the latest heuristics for solving it.

Let S be a set of 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 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 [5], constructing the optimal labeling of the internal nodes of a fixed tree T can be done in polynomial time [4].

2.1 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)* [9]. In TBR, an edge is removed from the given tree T and each pair of edges touching each endpoint merged, 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.

The *Parsimony Ratchet* [13] is an iterative technique that combines TBR with an interesting approach to move out of local optima. When a local optimum has been reached, i.e., when no further improvement can be made through a TBR move, the input data are modified by randomly doubling $p\%$ of the sites to produce new sequences that are $1.p$ times longer than the original input sequences. (Typically, p is 0.25.) Iterative improvement with TBR is then attempted on the new data. When this new search reaches a local optimum, the additional sites are removed (reverting to the original sequence length) and iterative improvement is resumed from this new configuration. The parsimony ratchet is implemented in two software packages, TNT [6] and PAUP* [20]. TNT provides a faster implementation, but unlike PAUP*, it is not publicly available; neither package is open-source.

2.2 Disk-Covering Methods

Disk-Covering Methods (DCMs) [7, 8, 12, 16, 21] 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 [21]. The second DCM [8], also called DCM2, was designed to speed up heuristic searches for MP trees.

3 Rec-I-DCM3

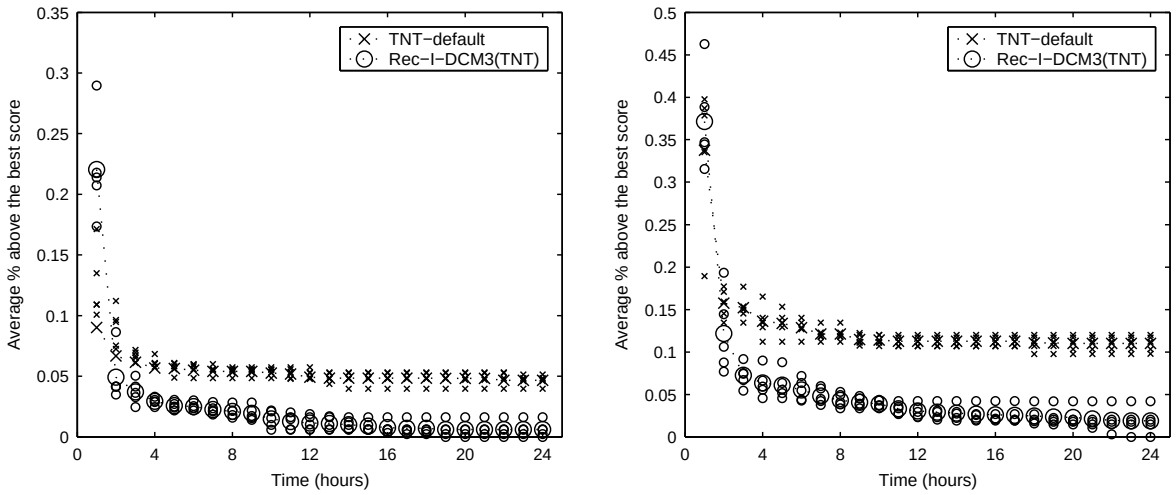
DCM1 can be viewed, in rough terms, as attempting to produce overlapping clusters of taxa to minimize the intracluster diameter; it produces good subproblems (small enough in size), but the structure induced by the decomposition is often poor. DCM2 computes a fixed structure (a graph separator) to overcome 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 allows 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, DCM3 was designed 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.

Roshan *et al.* designed DCM3 in part to avoid producing large subsets, as DCM2 is prone to do [17]. Yet, of course, the subproblems produced from a very large dataset remain too large for immediate solution by a base method (a phylogenetic tree reconstruction method of choice). Hence they used DCM3 recursively, producing smaller and smaller subproblems until every subproblem was small enough to be solved directly. In [17], Roshan *et al.* showed that DCM3 produced subproblems of sizes bounded by about half the initial subproblem size and much smaller than those produced by DCM2. (`Rec-I-DCM3` in that series of tests was set up to recurse until each subproblem was of size at most one quarter of the original size.)

Once the dataset is decomposed into overlapping subsets $A_1, A_2, \ldots, A_m$ ($m \leq 4$ 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 `Rec-I-DCM3` method [17] takes as input the set $S = \{s_1, \ldots, s_n\}$ of n aligned biomolec-

ular sequences, the chosen base method, and a starting tree T. In [17], the authors used TNT (with default settings) as the base method, since it is the hardest to improve (in comparison, the PAUP* implementation of the parsimony ratchet [2] is easier to improve). The Rec-I-DCM3 method produces smaller subproblems by recursively applying the centroid-edge decomposition until each subproblem is of size at most k. The subtrees are then computed, merged, and resolved (from the bottom-up, using random resolution) to obtain a binary tree on the full dataset. These steps are repeated for a specified number of iterations. Figure 1 demonstrates the significant improvement over TNT gained by employing the Rec-I-DCM3 booster to the method.



(a) The European RNA dataset (b) The RDPII dataset

Figure 1: (a)Average MP scores of TNT and Rec-I-DCM3(TNT) on the European RNA dataset, given as the percentage above the best score. Shown 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. (b) Average MP scores of TNT and Rec-I-DCM3(TNT) on the RDPII dataset, 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.

4 Parallel Rec-I-DCM3

As described in section 3, Rec-I-DCM3 is a divide-and-conquer algorithm, which makes it a natural candidate for parallelization. For the datasets we experimented with (up to 14000 taxa), the problem fits into memory, which simplifies the implementation. In Figure 2 we present a typical problem decomposition induced by Rec-I-DCM3; the decomposition contains the main problem, composite subproblems (C1, C2) and leaf subproblems (L1-L5).

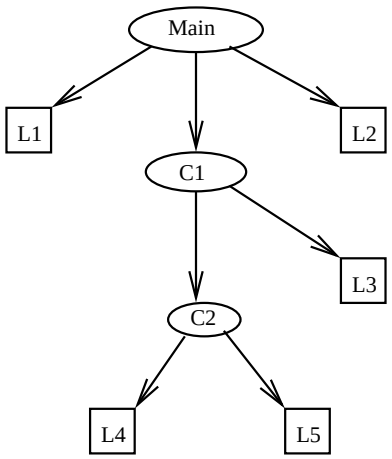


Figure 2: The `Rec-I-DCM3` decomposition.

Allen and Kennedy [1] discuss key requirements for high performance of parallel programs, such as load balancing and effi cient communication. The natural implementation for `PRec-I-DCM3` is to use task-parallelism with a master-slave model. The master node maintains a database of subproblems: subproblems available for solving ("available"), already solved subproblems ("solved"), and subproblems currently being solved by a worker ("active"). During its lifetime, a subproblem changes states from "available" to "active" to "solved". The master coordinates the distributed computation and ensures that the system is in a consistent state throughout the computation.

At the beginning of a `PRec-I-DCM3` iteration, the master performs a one-level decomposition of the main problem, using the current guide tree (see Section 3 for details regarding the use of a guide tree). Among the resultant subproblems, there are usually both leaf subproblems and composite ones. Next, it dispatches available subproblems, in decreasing order of their sizes, to the idle workers; the rationale is that the largest subproblems would take longer time to solve. Once a problem is dispatched, its state is changed from "available" to "active".

The worker processes wait for subproblems from the master. If the problem received is a leaf subproblem, then the worker invokes a standalone solver, such as `TNT` or `PAUP*`, and then returns the resulting subtree to the master. The solver runs without a time limit, because the leaf subproblems are small (usually not exceeding 2000 taxa). If the problem received is a composite subproblem, the worker decomposes it further into subproblems. It selects the largest subproblem among these, performs additional work on it, and then returns the remaining subproblems to the master.

When the master receives the solution for a leaf subproblem, it changes its state from "active" to "solved". It checks if all the children subproblems with the same parent as the current subproblem are solved; if this is the case, then it sends all the children subproblems to a worker for merging.

When the master receives the decomposition of a composite subproblem, it adds all the children subproblems to the problem database, with the state "available", and marks the subproblem kept by the worker as "active".

When a worker receives a command to merge a composite subproblem, it receives the solutions

for the children subproblems from the master and applies the strict consensus merging [7, 8], followed by a random refinement of the resulting tree. Finally, the solution is sent to the master.

When all subproblems of the main problem are solved, the master merges the their solution trees, then signals the workers to perform the random refining followed by the global search phase. These random refinements are done independently and usually result in different trees since each worker process has an independent random number generator. After the refinement phase, workers invoke the standalone solver to compute the parsimony score of the entire tree. The time of the search is limited to a fixed value (referred to as the *Global Search Time Limit*, or *GSTL*), which is provided at the program launch. The limit is necessary because the full-size problem is large (up to 14000 taxa). After finishing the global search phase, each worker sends the resultant parsimony scores to the master, which, in turn, selects the minimum score and retrieves the corresponding tree from the worker. This tree is used as the guide tree for the following iteration.

A significant advantage of `PRec-I-DCM3` over `Rec-I-DCM3` is that the former is able to run several instances of the global search phase in parallel. These instances start from different points in the tree space, potentially leading to a wider coverage of the tree search space and an alternate option for escaping local optima.

The current master-slave scheme implementation is a prototype. For very large datasets (20000+ taxa), the master can become a bottleneck flooded with incoming and outgoing communication traffic. A distributed master scheme will solve this problem. An emerging family of global address languages, such as Co-Array Fortran [14] or Unified Parallel C [3], might prove efficient for such an implementation because they provide one-sided communication on the language level perfectly suited for the distributed master scheme.

5 Experimental Settings and Results

The platform we used for experiments was a cluster of 92 HP zx6000 workstations interconnected with Myrinet 2000. Each workstation node contains two 900MHz Intel Itanium 2 processors with 32KB/256KB/1.5MB of L1/L2/L3 cache, 4GB of RAM, and the HP zx1 chipset. Each node is running the Linux operating system (kernel version 2.4.18-e plus patches). We used the Intel C/C++ compiler version 8.1 for Itanium.

We ran both `Rec-I-DCM3` and `PRec-I-DCM3` on the two largest datasets used in [17]:

- European RNA: a dataset of 11,361 aligned small subunit ribosomal Bacteria RNA sequences (1,360 sites) [23].
- RDPII: a dataset of 13,921 aligned 16s ribosomal Proteobacteria RNA sequences (1,359 sites) [10].

We report the average results of the methods over five runs on the two datasets.

Since `Rec-I-DCM3` uses strict consensus merging with random refinement, a good random number generator is essential to obtain credible results. Each node used the UNIX `random` random number generator initialized with a seed read from `/dev/random` at the beginning of a run.

We investigated two main questions:

1. What is the optimal choice of GSTL and subproblem size that yields the best results of `Rec-I-DCM3`?

2. Using the optimal choice of parameters, how does `PRec-I-DCM3` perform compared to `Rec-I-DCM3`?

To answer the first question, we ran `Rec-I-DCM3` on the European RNA and RDPII datasets for 13 hours, and recorded the average best scores obtained by the method for various subproblem sizes and GSTL values. Tables 1 and 2 show the average best scores obtained by `Rec-I-DCM3` on

Table 1: Average best scores obtained by `Rec-I-DCM3` on the European RNA dataset, with different GSTL values and subproblem sizes.

Max. Subproblem Size \ GSTL	4 min	8 min	16 min	32 min	60 min
500	273688	272326	272209	272160	272163
1000	272865	272194	272158	272163	272145
2000	272133	272127	272155	272133	272138
4000	272151	272146	272209	272154	272155

Table 2: Average best scores obtained by `Rec-I-DCM3` on the RDPII dataset, with different GSTL values and subproblem sizes.

Max. Subproblem Size \ GSTL	4 min	8 min	16 min	32 min	60 min
500	243545	242005	241131	241093	241088
1000	242529	241275	241140	241069	241042
2000	241054	241044	241062	241017	241068
4000	241135	241154	241118	241131	241097

the European RNA and RDPII datasets, respectively. The tables show that maximum subproblem size of 2000 taxa yields the best results on both datasets, under the conditions of our experiments. However, a GSTL of 8 minutes gave the best results on the European RNA dataset, while a GSTL of 32 minutes gave the best results on the RDPII dataset.

Further, we focused on maximum subproblem size of 2000 taxa, and investigated the performance of `Rec-I-DCM3` for different values of GSTL for the duration of 13-hour runs. Figures 3 and 4 show the results. These figures show that a GSTL of 8 minutes becomes consistently the optimal choice on the European RNA dataset after about 8 hours and 40 minutes, whereas a GSTL of 32 minutes becomes consistently the optimal choice on the RDPII dataset after about 6 hours and 30 minutes.

These results demonstrate that different datasets may require different parameter settings of `Rec-I-DCM3` in order to achieve the best performance. We expect that the choice of these parameters depends on the quality (in terms of parsimony score and topology) of the tree used as the

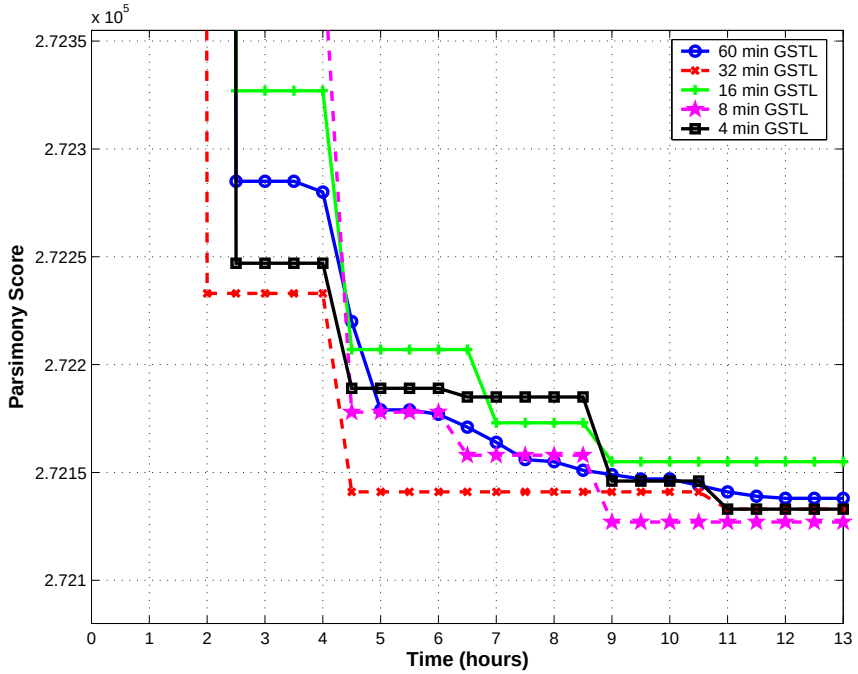


Figure 3: Results obtained by `Rec-I-DCM3` on the European RNA dataset, with maximum subproblem size of 2000 taxa and different GSTL values for global search.

start point in each iteration, as well as the evolutionary diameter[1] of the dataset. We will investigate this in our future work.

After determining the optimal choice of maximal subproblem size and GSTL values, we used these values for `PRec-I-DCM3` and compared its performance to that of `Rec-I-DCM3` under the same settings. The results are shown in Figures 5—8.

To compare the performance of `PRec-I-DCM3` and `Rec-I-DCM3`, we ran both methods on the two datasets for 13 hours, using a maximal subproblem size of 2000 taxa, and GSTL values of 8 and 32 minutes for the European RNA and RDPII datasets, respectively. We plotted the average parsimony score obtained by the two method as a function of time, and the topological difference between the best trees computed by the two methods as computed by the Robinson-Foulds (RF) metric [15] of topological tree difference. We now briefly review the RF metric.

Let T be an unrooted tree leaf-labeled by a set S of taxa. An edge $e = (u, v)$ in T defines a bipartition of S (the set of all leaves on one side of the edge, and the set of all other leaves). Let $C(T)$ be the set of bipartitions defined by all edges in tree T. The RF measure between two trees T and T' is defined as

$$RF(T, T') = \frac{\frac{|C(T)-C(T')|}{|C(T)|} + \frac{|C(T')-C(T)|}{|C(T')|}}{2}.$$

Figure 5 shows that, on the European RNA dataset, `PRec-I-DCM3` consistently outperforms

[1]The evolutionary diameter of a dataset is defined as the maximum number of changes between any two taxa in the dataset.

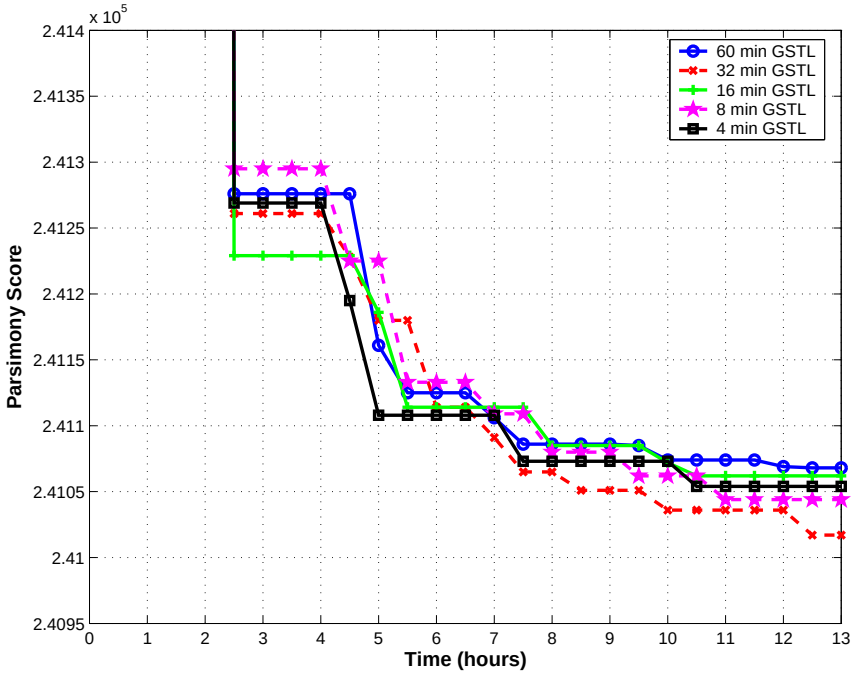


Figure 4: Results obtained by `Rec-I-DCM3` on the RDPII dataset, with maximum subproblem size of 2000 taxa and different GSTL values for global search.

`Rec-I-DCM3`, with the exception of the 2-CPU case. The figure shows that the best parsimony score computed by `Rec-I-DCM3` after 6 and a half hours is computed by `PRec-I-DCM3` after two and a half hours only using 8 or 16 workers. Further, the best parsimony score computed by `Rec-I-DCM3` after a complete run of 13 hours is computed by `PRec-I-DCM3` (using 8 and 16 workers) after only 7 and a half hours. Finally, despite a seemingly small difference in the parsimony scores computed by the two methods after 13 hours, Figure 6 shows that the actual trees computed by the methods differ in about 25% of their internal edges, according to the RF metric. This results shows a significant difference between the trees computed by the two methods on the European RNA dataset.

More dramatic improvements were observed on the RDPII dataset, as Figure 7 demonstrates. On this dataset, the performance of `PRec-I-DCM3` is consistently better than that of `Rec-I-DCM3`, regardless of the number of workers used in the `PRec-I-DCM3` implementation. The best performance of `PRec-I-DCM3` on this dataset is achieved using 8 and 16 worker CPUs, with a slight edge for the 8-CPU implementation after 11 hours. Notice that the best parsimony score computed by `Rec-I-DCM3` after the complete run of 13 hours is obtained by `PRec-I-DCM3` using 8 CPUs after only 4 hours. Figure 8 demonstrates that the difference between the parsimony scores obtained by the two methods after 13 hours translates into a 40% difference in the topologies (specifically, numbers of internal edges) of the trees computed by the two methods, according to the RF measure.

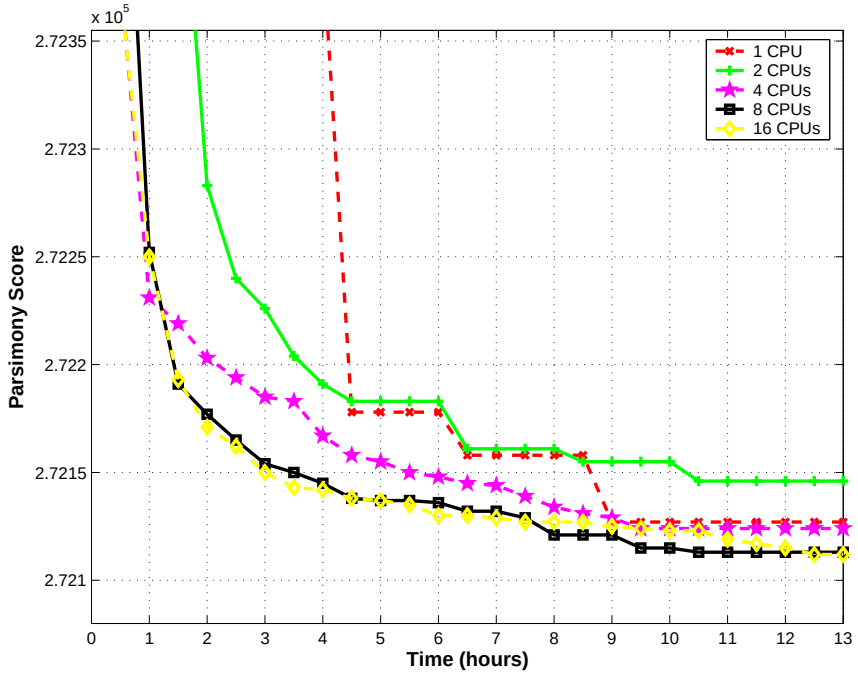


Figure 5: Results obtained by `Rec-I-DCM3` and `PRec-I-DCM3` on the European RNA dataset, with maximum subproblem size of 2000 taxa and GSTL of 8 minutes for global search. The 1-CPU curve corresponds to the `Rec-I-DCM3`, whereas the other curves correspond to `PRec-I-DCM3` using different numbers of CPUs.

6 Conclusions and Future Work

The `Rec-I-DCM3` method of Roshan *et al.* was the first technique that allowed a successful application of parsimony heuristics with high accuracy within reasonable time limits. Nonetheless, in order to reconstruct, with high accuracy, phylogenetic trees at a much larger scale, further speed-up and improvements are imperative. In this paper we introduced the first such improvement through `PRec-I-DCM3`, a parallel version of the `Rec-I-DCM3` method. We implemented and ran `PRec-I-DCM3` on two large datasets. The results demonstrated a significant improvement over `Rec-I-DCM3`.

Directions for future work include:

- Exploring a distributed master scheme.
- Investigating the difference in optimal parameter choice across different datasets.
- Experimental testing of `PRec-I-DCM3` on simulated datasets. Using simulations allows for investigating the performance of the method with respect to the "true" tree, which is known in such studies (as opposed to real datasets, in which the true tree is not known).
- Existing implementations of `TNT` and `PAUP*` are limited to handle up to 16,000-taxon trees.

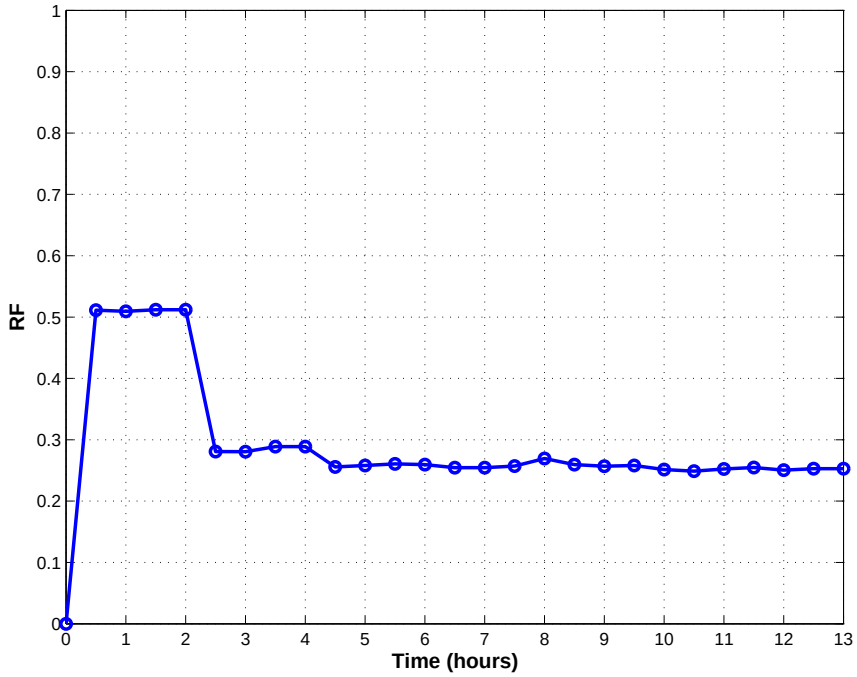


Figure 6: RF values between the best trees obtained by `Rec-I-DCM3` and `PRec-I-DCM3` (on 8 CPUs) on the European RNA dataset.

We intend to study the performance of `PRec-I-DCM3` on datasets larger than the ones we used, once tools that handle more than 16,000 taxa are available.

- Application of `Rec-I-DCM3` and `PRec-I-DCM3` to likelihood heuristics.

7 Acknowledgments

The authors would like to thank Erion Plaku for helpful discussions, and Derek Ruths for providing us with the code for computing the Robinson-Foulds distance between trees.

References

[1] Randy Allen and Ken Kennedy. *Optimizing Compilers for Modern Architectures: A Dependence-Based Approach*. Morgan Kaufmann Publishers, San Francisco, CA, 2001.

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

[3] W. W. Carlson, J. M. Draper, D. E. Culler, K. Yelick, and K. Warren E. Brooks. Introduction to UPC and language specifi cation. Technical Report CCS-TR-99-157, IDA Center for Computing Sciences, May 1999.

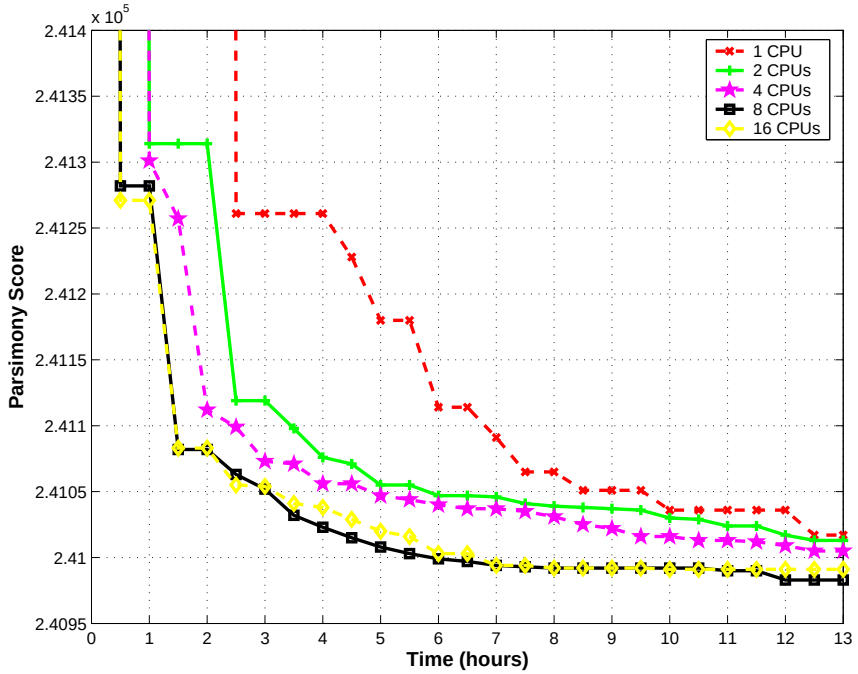


Figure 7: Results obtained by `Rec-I-DCM3` and `PRec-I-DCM3` on the RDPII dataset, with maximum subproblem size of 2000 taxa and GSTL of 32 minutes for global search. The 1-CPU curve corresponds to the `Rec-I-DCM3`, whereas the other curves correspond to `PRec-I-DCM3` using different numbers of CPUs.

[4] W.M. Fitch. Toward defi ning the course of evolution: minimum change for a specifi ed tree topology. *Syst. Zool.*, 20:406–416, 1971.

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

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

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

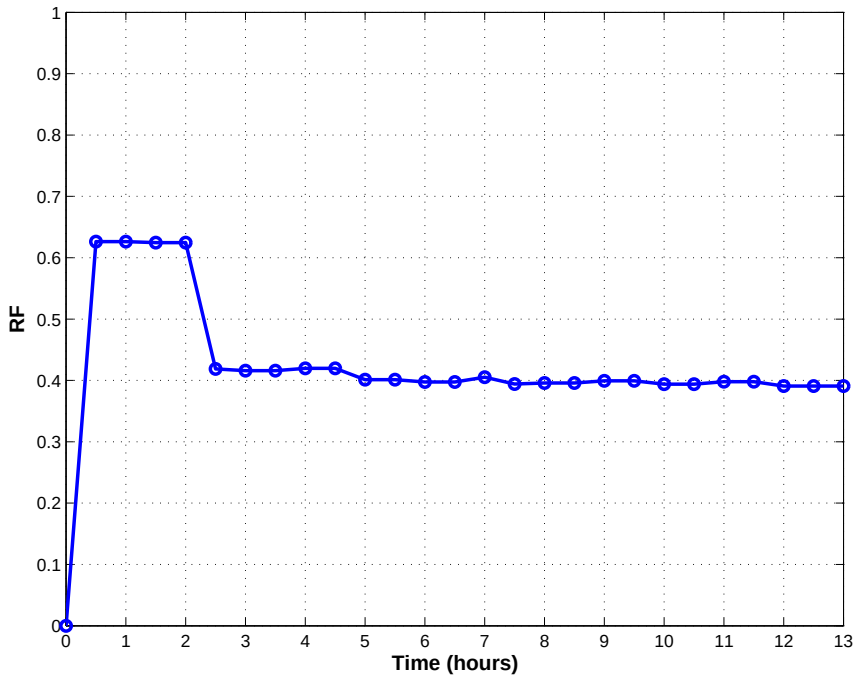


Figure 8: RF values between the best trees obtained by `Rec-I-DCM3` and `PRec-I-DCM3` (on 8 CPUs) on the RDPII dataset.

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

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

[12] 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.

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

[14] R. W. Numrich and J. K. Reid. Co-Array Fortran for parallel programming. Technical Report RAL-TR-1998-060, Rutheford Appleton Laboratory, August 1998.

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

[16] 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 Su-*

pertrees: Combining Information to Reveal the Tree of Life, volume 3 of *Computational Biology*, pages 301–328. Kluwer Academic Publishers, 2004.

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

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

[19] 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.

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

[21] 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.

[22] T.L. Williams, B.M.E. Moret 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.

[23] 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.