Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB 2007)

Reconsidering the Performance of Cooperative Rec-I-DCM3

Tiffani L. Williams
Department of Computer Science
Texas A&M University
College Station, TX 77843
Email: tlw@cs.tamu.edu

Marc L. Smith
Computer Science Department
Vassar College
Poughkeepsie, NY 12604
Email: mlsmith@cs.vassar.edu

***Abstract*— Phylogenetic heuristics attempt to find the best trees within a limited amount of time in an exponentially-sized tree space. It is believed that better-scoring trees are a good approximation of the "true" evolutionary history of a set of organisms. Previous work by the authors investigated the usefulness of cooperative heuristics as an efficient parallel algorithm. Cooperative heuristics are similar in spirit to evolutionary algorithms since they use a population of diverse solutions to search an exponentially-sized tree space for better-scoring (and thus more accurate) trees.**

In this paper, we study whether cooperation can lead to a competitive sequential algorithm for inferring phylogenetic trees quickly and accurately. The two algorithms under investigation are Rec-I-DCM3, one of the best maximum parsimony algorithms available for reconstructing phylogenies, and its cooperative counterpart, Cooperative Rec-I-DCM3. We compare their convergence rates to best-known scores on two datasets consisting of 921 and 2,000 taxa. Instead of strictly relying on wall-clock time, we devise a new performance measure called *tree performance*, which provides more robust comparisons across different architectures and implementations. Under both measures, Cooperative Rec-I-DCM3 is a competitive sequential algorithm as it outperforms Rec-I-DCM3 on both datasets.

I. Introduction

One of the grand challenges facing biology is assembling "The Tree of Life,", the evolutionary history of all-known organisms. Science and society would benefit enormously from detailed and accurate knowledge of the phylogenetic relationships among the world's organisms. Unfortunately, we cannot know the true evolutionary history of a set of organisms (or taxa), so the problem is often reformulated as an optimization problem. Under the criterion of interest, trees are given a score, and better scoring trees are assumed to be better approximations of the truth. For n taxa, the number of possible hypotheses (or trees) is $(2n-5)!!$. It is infeasible to consider all possible explanations for any moderately-sized dataset (> 30 taxa). Instead, phylogenetic heuristics attempt to find the best possible trees within a limited amount of time.

In this paper, we reevaluate the performance of Cooperative Rec-I-DCM3 [1], [2] as a competitive sequential algorithm that uses a *memetic-based*, computational-intelligence approach [3] to infer evolutionary trees. Cooperative algorithms use a diverse population of solutions to guide the search for the best-scoring maximum parsimony (MP) trees. Under MP, the tree that explains the data with the fewest evolutionary events (i.e., mutations) is the one that is preferred. MP is an NP-hard problem [4], but the problem of scoring a fixed tree is polynomial [5]. Previous work by the authors shows that cooperation worked quite well as a parallel approach [1], [2]. However, we were curious whether cooperation can lead to an improved sequential algorithm. Thus, our results are meant to show that cooperation—although an embarrassingly parallel approach—can be used as a general-purpose technique for the development of sequential and parallel phylogenetic heuristics.

Here, we study the sequential performance of Cooperative Rec-I-DCM3 and its non-cooperative counterpart, Rec-I-DCM3 [6]. Rec-I-DCM3 is the newest member of the family of Disk-Covering Methods (DCMs) [7], [8], [9], [6], which use a divide-and-conquer approach to reconstructing evolutionary trees. Experimental results have shown that Rec-I-DCM3 performs quite well over a large range of datasets. Given that good performance from a search algorithm is due to a balance of *exploration* (the generation of new tree solutions in untested regions of the search space) and *exploitation* (the concentration of the search in the vicinity of known good solutions), it is unclear whether both of these objectives can be met by search algorithms such as Rec-I-DCM3 that manipulate a single tree. On the other hand, a population-based approach such as Cooperative Rec-I-DCM3 seems better adapted to balance both of these objectives.

In this study, algorithms are compared in terms of their *convergence rate* to best-known scores on two biological datasets consisting of 921 and 2,000 taxa. Of particular interest to us is the convergence rate of the algorithms as both a function of time (wall-clock performance) and number of trees searched (tree performance). *Tree performance* is a new measure that provides an architecture- and implementation-independent assessment of an algorithm's convergence rate. One benefit of this measure is that an algorithm is not penalized because of its implementation. Thus, wall-clock performance and tree performance provide a more balanced view of a search algorithm's performance. Our results demonstrate that Cooperative Rec-I-DCM3 is a competitive sequential algorithm. The improvement is impressive; it is the best-overall performer in terms of wall-clock performance and tree performance on both biological datasets studied here.

II. Maximum Parsimony and Heuristic Performance

Maximum parsimony (MP) is an optimization problem for inferring the evolutionary history of different taxa, in which it is assumed that each of the taxa in the input is represented by a string over some alphabet. The symbols in the alphabet can represent nucleotides (in which case, the input are DNA or RNA sequences), or amino-acids (in which case the input are protein sequences), or may even include discrete characters for morphological properties. It is also assumed that the strings are put into a multiple alignment, so that they all have the same length. Maximum parsimony then seeks a tree, along with inferred ancestral sequences, so as to minimize the total number of evolutionary events (counting only point mutations).

A. Formal definition of maximum parsimony

Formally, given two sequences a and b of the same length, the *Hamming distance* between them is defined as $|\{i : a_i \neq b_i\}|$ and denoted as $H(a, b)$. Let T be a tree whose nodes are labeled by sequences of length k, and let $H(e)$ denote the Hamming distance of the sequences at each endpoint of edge e. The *parsimony length* of the tree T is $\sum_{e \in E(T)} H(e)$. The MP problem seeks the tree T with the minimum length; this is the same as seeking the tree with the smallest number of point mutations for the data. MP is an NP-hard problem [4], but the problem of assigning sequences to internal nodes of a fixed leaf-labelled tree is polynomial [5].

B. Measuring the performance of MP heuristics

Performance studies of MP heuristics have generally focused on speed. Hence, studies that explore speed have examined how much running time each heuristic can solve MP (or reach the current best known score) for specific real biological datasets (see [10], [6], [2], [11] for examples of such studies). Maximum parsimony searches are based on the assumption that better scoring trees are more accurate approximations of the true evolutionary relationships between a set of organisms. Thus, heuristics that converge quickly (in terms of running time) to the best-known score are of most interest to the phylogenetic community.

However, relying simply on running time to judge the merits of a phylogenetic heuristic has several limitations. In particular, running time (or wall-clock performance) is neither an architecture- nor implementation-independent measure of performance. For example, suppose we are interested in comparing the running time of two phylogenetic heuristics A and B. If heuristic A requires less running time to converge to the best-known score than heuristic B, it is unclear whether A is algorithmically superior or better implemented than B. Ideally, we would prefer A's dominance to be a result of algorithmic superiority and not programming tricks. We cannot know why A is better than B if we rely strictly on running time as a performance measure.

Besides running time, additional performance measures are needed to provide an overall assessment of a phylogenetic heuristic. In this paper, we devise a new measure called *tree performance* that when combined with running time provides a more informative assessment of heuristic performance. Here, tree performance measures the convergence rate of a phylogenetic heuristic as function of the total number of output trees returned from the search (see Section VI-B).

Ultimately, better performance measures lead to better assessments of phylogenetic heuristics. Moreover, these improved measures will motivate the design of better phylogenetic heuristics, which lead to more accurate depictions of evolutionary relationships.

III. Rec-I-DCM3: A Single-Point Phylogenetic Heuristic

Recursive-Iteration DCM3 (Rec-I-DCM3) [6] is a single-point heuristic, for finding the best trees in tree space. Single-point refers to the way the heuristic achieves its goal, by working to improve a single solution. Most popular phylogenetic heuristics [10], [12], [13] manipulate a single solution at a time. As its name suggests, Rec-I-DCM3 implements a disk-covering method (DCM) [7], [8], [9], [6] to improve the score of the trees it finds. A DCM is a divide-and-conquer technique that consists of four stages: divide, solve, merge, and refine. At a high level, these stages follow directly from DCM being a divide-and-conquer technique.

During the divide phase, the input tree's leaf nodes, which contain the taxa, are divided into subproblems, or overlapping subsets of taxa. Each subset contains one or more taxa that can be found in another subset. (This overlapping property of the subsets becomes important for the merge phase.) Next, the DCM solves each of the subproblems, utilizing some existing phylogeny reconstruction technique. The result is a collection of phylogenies that correspond to the subsets of taxa from the divide stage of the DCM. Since each of these subproblem tree solutions overlap in the taxa they contain, they can be merged together using a consensus technique like strict consensus [7]. Strict consensus forms a new tree from two input trees by preserving the subtrees found in both input trees. This process is repeated until one tree remains. The preserving of subtrees during this process is what tends to produce multifurcations in the merge stage of the DCM, which is why the merged tree must be refined into a bifurcating tree.

Rec-I-DCM3, involves all of the above stages, but in addition, is both recursive and iterative. The recursive part concerns the divide stage of the DCM, where after dividing the input tree's leaf nodes into overlapping subsets of taxa, or subproblems, the subproblems themselves may be further divided into smaller subproblems. This is an important enhancement to the DCM approach since for very large datasets, the subproblems remain too large for an immediate solution. Thanks to the recursion, the subproblems are eventually small enough that they may be solved directly using some chosen base method. At this point, Rec-I-DCM3 uses strict consensus merger to do the work of recombining the overlapping subtrees to form a single tree solution. The iterative part of Rec-I-DCM3 refers to the repetition of the entire process just described. That is, the

resulting tree solution becomes the input tree for a subsequent iteration of Rec-I-DCM3.

Despite merging subtrees, it is important to remember that Rec-I-DCM3 is a single-point heuristic. The trees being merged are not multiple trees from tree space, since their leaf nodes are only a subset of the taxa from the original dataset. Thus, only one tree's solution is being evolved throughout Rec-I-DCM3's entire process. Each iteration represents one step in the evolution of its final solution; the input and output of each iteration is a single tree.

IV. Cooperative Rec-I-DCM3: A Population-Based Approach

Cooperative Rec-I-DCM3 [1], [2] leverages multiple tree solutions to to guide its search for better trees. Figure 1 provides a schematic diagram depicting the essential difference between the two algorithms studied here. Cooperative Rec-I-DCM3 consists of the following four steps.

1) Create a population of μ initial tree solutions.
2) For each of the μ trees, run Rec-I-DCM3.
3) Create a new tree population by performing selection and recombination on the trees from step 2.
4) Repeat steps 2 and 3 for the desired number of iterations.

The starting tree population can be created using any method such as random sequence addition [14]. In subsequent iterations, the starting tree population will come from the previous iteration's merged tree population. An iteration is divided into two phases (steps 2 and 3 above). In the first phase, a local search is performed on each tree in the starting tree population. Upon conclusion of the μ local searches, a new population of trees is ready for the next phase. During the second phase, trees are selected for inclusion into the merged tree population on the basis of their MP scores. The merged tree population serves as the starting tree population for the next iteration of the algorithm.

A. Selection and recombination

The selection process decides which solutions will enter the population of the next iteration. The μ trees from step 2 are ranked based on their MP scores, with the best scoring MP tree having the best rank. Next, the trees are placed into sets ($A, B,$ and C) based on their rank. The algorithm also keeps a list of trees with the best solution found by the current iteration of the search. These elite trees are placed into set A. The top-ranking trees from step 2 are placed into set B, and the lower-ranking trees are put into set C. These sets of trees comprise the new population. However, trees in set C may be recombined with any tree in $A, B,$ or C to create new (and more diverse) solutions. If $t \in C$ is chosen for recombination, it will be replaced by the resulting tree from the recombination phase.

For each tree $t \in C$, there is a $p\%$ chance that it will undergo recombination with a random tree $t' \in A \cup B$. t and t' are recombined by computing their strict consensus tree, which contains all of the bipartitions that are common between the trees. Since the strict consensus tree typically results in a multifurcating tree, it is refined into a binary tree and subjected to a global search using Tree-Bisection and Reconnection (TBR) [12]. In our experiments, trees in set A are comprised from the top 40% of trees found from the μ local searches. Moreover, the percentage of recombination was set to 25%.

B. Comparison to Rec-I-DCM3

A comparison of Cooperative Rec-I-DCM3 and Rec-I-DCM3 shows that both algorithms are examples of divide-and-conquer strategies designed to boost the performance of existing algorithms. In fact, the use of populations in cooperative algorithms is a natural extension of subproblems in DCMs. However, the essential difference between the two approaches is that Rec-I-DCM3 uses a single tree to guide its search whereas Cooperative Rec-I-DCM3 incorporates a diverse population of solutions.

Also, both approaches employ the notion of decomposition and merging quite differently. The use of a population of trees in cooperative algorithms may be viewed as a partial decomposition of the exponentially large space of tree solutions. Each of the tree solutions represents a decomposition of that space. DCMs decompose the original problem of n taxa into a population of overlapping subsets. Hence, in DCMs, the population represents partial solutions to the original problem whereas in cooperative algorithms the population contains complete solutions. Moreover, the purpose of merging in both approaches is quite different. Cooperative algorithms use merging (or recombination) to create new, more diverse solutions. DCMs, on the other hand, require merging in order to obtain a single, complete solution on the entire dataset from the partial solutions.

Similarities to our cooperative approach is evident in other single-point heuristic used in phylogenetic software packages such as TNT [10]. Tree fusing and sectorial searches are very much in the spirit of our cooperative approach. Specifically, we could substitute the tree fusing algorithm into the recombination phase of our cooperative algorithm. Furthermore, one could have a collection of sectorial searches cooperate with each other to achieve greater performance.

C. Comparison to genetic algorithms

Our cooperative algorithms are analogous to techniques used in evolutionary algorithms. Indeed, we are using a memetic-based approach [3] to design a population-based heuristic. Memetic algorithms combine a population-based global search and an individual local search. While we are unaware of other phylogenetic techniques that use a memetic-based approach, there are several phylogenetic heuristics that employ genetic algorithms to search for phylogenetic trees based on maximum likelihood [15], [16], [17], [18].

The essential difference between our cooperative approach and genetic algorithms is how the approaches use an individual solution. In a typical genetic algorithm, individuals in the population are subjected to function evaluation followed by selection, recombination, and mutation. Each individual in the

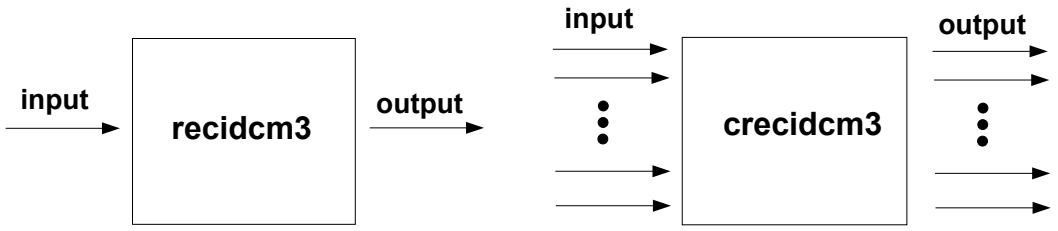


Fig. 1. A schematic depicting one iteration of the Rec-I-DCM3 and Cooperative Rec-I-DCM3 algorithms. Rec-I-DCM3 uses a single tree to guide its decisions. Cooperative Rec-I-DCM3, on the other hand, uses multiple trees (or a population) as its mechanism for finding better trees.

population is not subjected to an individual local search in a genetic algorithm approach.

In our Cooperative Rec-I-DCM3 algorithm, each solution in the population is the input to a local search algorithm, which is Rec-I-DCM3. However, we could easily use other local search algorithms such as parsimony ratchet [13]. The outputs of the Rec-I-DCM3 searches are the new individuals of the Cooperative Rec-I-DCM3 population, which is then subjected to selection and recombination. At present, our approach does not mutate the individual tree solutions.

V. Experimental Methodology

A. Datasets

Although simulated datasets can be used, phylogenetic researchers have noted that MP seems easier to solve on simulated data than on real data. Since simulated data is not sufficiently realistic, we chose to use biological datasets in our study. While our results are biologically relevant, we do not know the "true" evolutionary history of our data. Therefore, we use tree scores based on maximum parsimony as an approximation of the Our experimental results are based on the following two biological datasets.

1) A set of 921 aligned Avian Cytochrome b DNA sequences [19].
2) A set of 2,000 aligned Eukaryotic sRNA sequences (1251 sites) obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.

B. Parameter settings

Our overall objective is to study whether cooperation has a positive impact on sequential performance. We study this question by observing the behavior of Rec-I-DCM3 and Cooperative Rec-I-DCM3. We set the parameters of each algorithm according to the recommended settings in the literature. For Rec-I-DCM3, it is recommended that the maximum subproblem size is 50% for datasets with 1,000 or less sequences and 25% for larger datasets not containing over 10,000 sequences. Hence, the maximum subproblem size for Dataset #1 and Dataset #2 is 461 and 500 taxa, respectively. We used the recommended settings established by Roshan et. al [6] for using TNT as a base method within the Rec-I-DCM3 algorithm.

The above parameter settings of Rec-I-DCM3 were also used when incorporated into the local search step of Cooperative Rec-I-DCM3. For our runs of Cooperative Rec-I-DCM3, we used a population size of eight. Cooperative algorithms also require selection and recombination parameter settings, which are described in Section IV-A. Finally, each algorithm requires the number of iterations to be specified. But, this varies in our experiments depending upon the performance measure of interest. We defer discussion of this parameter setting until Section VI.

C. Implementation and Platform

We used TCP Linda [20], an implementation of Gelernter's Linda [21] model of concurrency, to implement our cooperative algorithm. Although Cooperative Rec-I-DCM3 can take advantage of a parallel platform, we limited its execution to a single processor for the experiments in this paper. Our TCP Linda programs were written in the C-Linda language, which augments the C language with four primitive operations that permit process creation and access to tuple space — an associative, distributed shared memory. Rec-I-DCM3 is open-source software provided by Usman Roshan. TNT [10] was used as the base method for Rec-I-DCM3, and we used TNT's implementation of TBR. We used PAUP*'s [14] implementation of strict consensus.

Our experiments were performed on an Apple Workgroup Cluster for Bioinformatics, which consists of four, 64-bit, dual-processor nodes (eight total CPUs) with gigabit-switched interconnects. The cluster consists of Xserve G5 nodes, each of which contains two, 2 GHz PowerPC G5 processors. Each processor contains 512 KB of L2 cache and a 1 GHz front-side bus; the two processors on each node share 4 GB of DDR 400 MHz SDRAM (16 GB total RAM across the cluster).

VI. Experimental Results

Since our study is based on biological datasets, the true tree cannot be known precisely. We use MP scores as an approximation of the true tree. Let s_t represent the best score found by time t. Hence, we show performance in terms of number of steps $(s_t - b)$ from the best-known score, b, of a dataset. Of particular interest to us is an algorithm's *convergence rate* (in terms of number of steps) to the best-known score. Since Rec-I-DCM3 is a single-point heuristic, it produces a

Algorithm	Dataset #1		Dataset #2	
	it	tree	it	tree
Rec-I-DCM3	52.99	52.99	109.57	109.57
Cooperative Rec-I-DCM3	499.78	62.47	1449.07	181.13

TABLE I

THE RUNNING TIME (IN SECONDS) IS BROKEN INTO TWO COMPONENTS—THE TIME TO COMPLETE AN ITERATION AND THE TIME TO RETURN A SINGLE TREE (OR SOLUTION). FOR REC-I-DCM3, EACH ITERATION IS RESPONSIBLE FOR PROVIDING A SINGLE SOLUTION. UNDER COOPERATIVE REC-I-DCM3, EACH ITERATION PROVIDES EIGHT TREES. EACH VALUE IN THE TABLE IS THE AVERAGE OF THREE RUNS.

single tree every iteration. Our sequential runs of Cooperative Rec-I-DCM3, on the other hand, used a population of size eight, which is based on our previous experiments with the algorithm.

A. Wall-clock performance

Table I shows the running time required for each algorithm on our experimental platform. An iteration of Cooperative Rec-I-DCM3 requires more time to complete than Rec-I-DCM3. However, since Cooperative Rec-I-DCM3 outputs eight trees every iteration, the time to produce a single tree is comparable to the iteration time of Rec-I-DCM3. The addition of the selection and recombination accounts for the extra overhead in the Cooperative Rec-I-DCM3 algorithm.

However, even with this additional overhead, Figures 2(a) and 3(a) show the performance gain attained by using a cooperative approach. The total time shown in the plots is based on how long it takes Cooperative Rec-I-DCM3 to complete 100 iterations. Hence, in Figures 2(a) and 3(a), Cooperative Rec-I-DCM3 requires 13.88 and 40.25 hours to complete 100 iterations, respectively. During the same amount of time, Rec-I-DCM3 completes 944 and 1,324 iterations on Dataset #1 and Dataset #2, respectively. Under wall-clock performance, Cooperative Rec-I-DCM3 is a competitive sequential algorithm for phylogenetic search when compared to Rec-I-DCM3. It establishes itself as the best algorithm on both datasets within eight hours. Once the time limit is reached, Cooperative Rec-I-DCM3 is within eight and two steps of the best-known score for Dataset #1 and Dataset #2, respectively.

B. Tree-performance

Given that a phylogenetic analysis must complete in a reasonable amount of time, wall-clock performance is a very important performance measure. However, wall-clock performance captures any overhead that is associated with an algorithm. Hence, it is neither an architecture- nor implementation-independent measure of performance. Algorithms with poor implementations will always fair poorly under this measure. For such algorithms, it is unclear whether they are truly inferior algorithms or simply poorly-implemented ones.

One way to approach having an architecture- and implementation-independent performance measure is to represent the algorithms as black boxes where we only observe their input/output behavior (see Figure 1). Here, the total number of output trees serves as a substitute for time. Thus, we can measure the convergence rate of the algorithms as function of the total number of output trees returned. The CPU speed in which the trees are returned is ignored.

Figures 2(b) and 3(b) shows the performance of the algorithms in terms of their tree performance. Once again, Cooperative Rec-I-DCM3 outperforms Rec-I-DCM3. Hence, Rec-I-DCM3 has to consider more trees in order to potentially reach the performance levels of Cooperative Rec-I-DCM3. However, the most striking feature of the plots is how well tree performance mimics the wall-clock performance curves. The plots correlates well with the fact that each algorithm records its execution time at the end of each iteration.

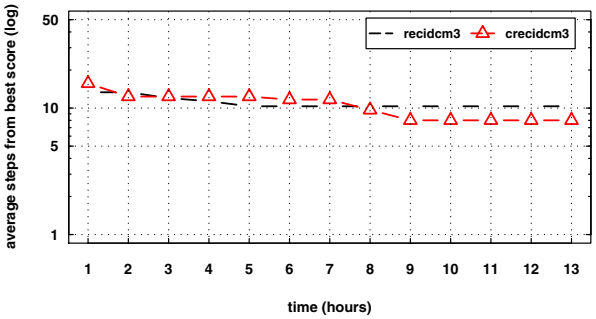
Thus, nothing is lost by looking at the convergence rates of the algorithms in terms of their tree performance. However, the benefits of having such a measure is significant. In particular, it allows one to compare the essence of the algorithms, which may be hidden because of the way an algorithm is implemented. For our cooperative algorithm, both measures show that there is essentially no penalty for using cooperation as an enhancement to the Rec-I-DCM3 algorithm. Furthermore, the tree performance measure shows that Cooperative Rec-I-DCM3's good overall performance is not a result of implementation heroics or parallel execution.

The use of the tree performance measure also implications for parallel phylogenetic heuristics. Parallelization does not improve the tree performance of a phylogenetic heuristic. However, running time is improved. Thus, parallelization allows heuristics to produce an output of t trees faster than can be done sequentially.

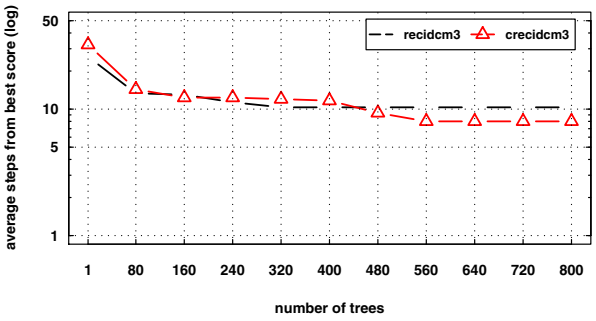
C. Distribution of tree scores

Next, we look at the distribution of the tree scores for both datasets and the two heuristics being compared: Rec-I-DCM3 and Cooperative Rec-I-DCM3. For each dataset and heuristic, over the course of three experimental runs, 2,400 total trees were returned (800 trees per run). Figures 4 and 5 provide the respective histograms of the MP scores for each dataset, by Rec-I-DCM3 and Cooperative Rec-I-DCM3. Histograms provide a visual representation of where an algorithm spends its time in terms of MP scores. Each step along the x-axis of the histograms represents a subset of trees with the same tree score.

For both datasets, Cooperative Rec-I-DCM3 finds more trees with MP scores closer to each respective dataset's best scores than does Rec-I-DCM3. In the case of Dataset #1, the smaller dataset, the subsets of trees are similar in terms of MP scores and steps above the best score. Even so, the histograms reveal Cooperative Rec-I-DCM3 finds better trees overall. For example, Cooperative Rec-I-DCM3 finds more trees than Rec-I-DCM3 within 20 steps of the best score, and fewer trees than Rec-I-DCM3 over 40 steps from the best score. Dataset #2, the larger dataset, tells a dramatically different story. Almost all of the trees found by Cooperative Rec-I-DCM3 are within 20 steps of the best score for Dataset #2, and about half of

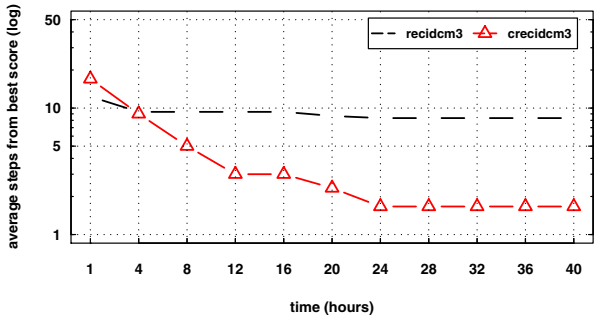


(a) Wall-clock performance

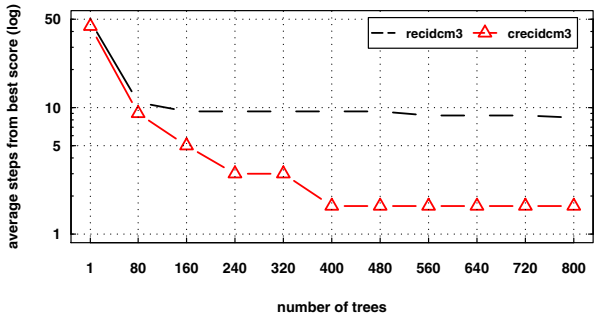


(b) Tree performance

Fig. 2. Performance of the algorithms on Dataset #1 (921 taxa). Each data point represents the average best score found over three runs. The best score is 40,494. (a) The time limit was established by how long it took for Cooperative Rec-I-DCM3 to complete 100 iterations. In the same amount of time, Rec-I-DCM3 completed 944 iterations. (b) With eight solutions per iteration for Cooperative Rec-I-DCM3, 800 trees is equivalent to 100 iterations. Rec-I-DCM3 required 800 iterations to output 800 trees.



(a) Wall-clock performance



(b) Tree performance

Fig. 3. Performance of the algorithms on Dataset #2 (2,000 taxa). Each data point represents the average best score found over three runs. The best score is 74,534. (a) The time limit was established by how long it took for Cooperative Rec-I-DCM3 to complete 100 iterations. In the same amount of time, Rec-I-DCM3 completed 1,324 iterations. (b) With eight solutions per iteration for Cooperative Rec-I-DCM3, 800 trees is equivalent to 100 iterations. Rec-I-DCM3 required 800 iterations to output 800 trees.

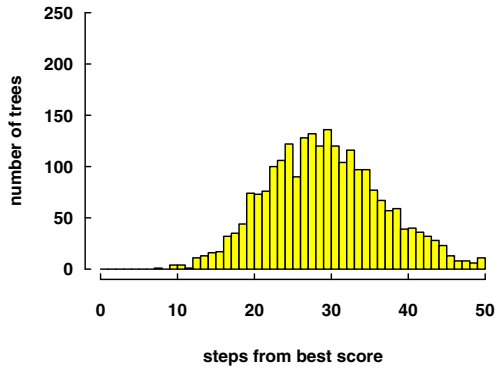
those are within 10 steps of the best score. For Rec-I-DCM3, almost all of the trees found are within 30 steps of the best score, with fewer than half of those trees within 15 steps of the best score.

VII. Conclusions and Future Work

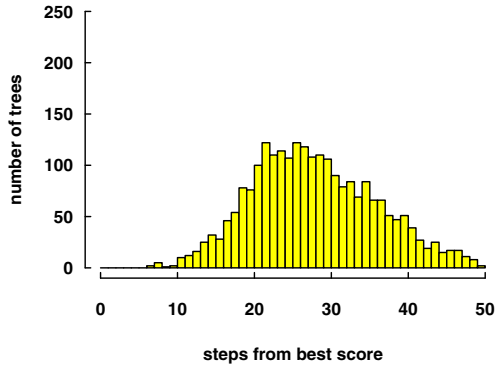
Our study clearly shows the improvement that results from placing Rec-I-DCM3 in a cooperative framework. On both biological datasets, it outperforms Rec-I-DCM3 in terms of wall-clock and tree performance. While we consider tree performance a more neutral measure of performance, other studies have considered counting the number of branch swap operations (e.g., TBR moves) as a measure of computational effort. Although branch-swapping is a fundamental operation in phylogenetic search, the composition of phylogenetic heuristics is more than a series of TBR operations. For example, some steps of the Rec-I-DCM3 algorithm do not require branch-swapping (e.g., problem decomposition and merging subproblems). So, counting branch-swaps wouldn't necessarily measure the influence of such steps on Rec-I-DCM3's performance.

Ultimately, what matters the most in a phylogenetic search is the quality of the inferred tree. The tree performance measure treats the phylogenetic heuristic as a black box and simply measures its output behavior on an iterative basis. This measurement doesn't penalize an algorithm for the amount of time it takes to produce a tree. The tree performance measure simply measures the worthiness of the output regardless of the time required to produce it. Of course, in the end, the time required to produce good trees does matter. Thus, tree performance coupled with wall-clock performance provides a more balanced assessment of the empirical performance of a search heuristic. Under both of these measures, the cooperative approach substantially improves the Rec-I-DCM3 algorithm.

For future work, we plan to continue studying the implications of using tree performance (as well as other implementation- and architecture-independent measures) as a substitute for running time. In particular, we are interested in

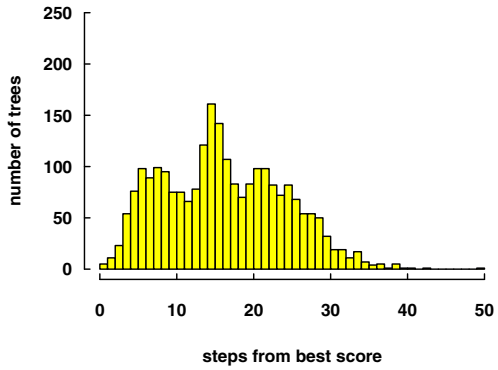


(a) Rec-I-DCM3

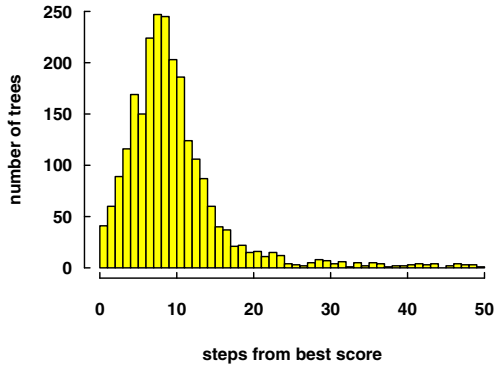


(b) Cooperative Rec-I-DCM3

Fig. 4. Distribution of tree scores on Dataset #1 (921 taxa). Each run of the algorithm returned 800 trees. Over the course of three runs, there were a total of 2,400 trees, which is the number of trees reflected in the histograms.



(a) Rec-I-DCM3



(b) Cooperative Rec-I-DCM3

Fig. 5. Distribution of tree scores on Dataset #2 (2,000 taxa). Each run of the algorithm returned 800 trees. Over the course of three runs, there were a total of 2,400 trees, which is the number of trees reflected in the histograms.

applying the measure to other heuristics such as parsimony ratchet [13]. Although this paper focused on cooperative search algorithms for maximum parsimony, our methodology should be equally applicable to maximum likelihood. Hence, our future plans include applying our approach to improve the convergence rates of maximum likelihood searches. Finally, our study focused on evaluating phylogenetic trees based on their MP scores. Thus, we also plan to evaluate search algorithms in terms of the distances between the tree topologies found.

VIII. Acknowledgments

The authors would also like to thank Usman Roshan for providing the code for Rec-I-DCM3 and the datasets used for this study.

References

[1] M. L. Smith and T. L. Williams, "Phylospaces: Reconstructing evolutionary trees in tuple space," in *Proc. Fifth IEEE International Workshop on High Performance Computational Biology (HiCOMB 2006)*, April 2006. [Online]. Available: http://www.hicomb.org/papers/HICOMB2006-03.pdf

[2] T. L. Williams and M. L. Smith, "Cooperative-Rec-I-DCM3: A population-based approach for reconstructing phylogenies," in *Proc. Third IEEE Symp. on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB'05)*, 2005, pp. 127–134.

[3] P. A. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," Caltech, Tech. Rep., 1989.

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

[5] W. M. Fitch, "Toward defining the course of evolution: minimal change for a specific tree topology," *Syst. Zool.*, vol. 20, pp. 406–416, 1971.

[6] U. Roshan, B. M. E. Moret, T. L. Williams, and T. Warnow, "Rec-I-DCM3: a fast algorithmic techniques for reconstructing large phylogenetic trees," in *Proc. IEEE Computer Society Bioinformatics Conference (CSB 2004)*. IEEE Press, 2004, pp. 98–109.

[7] D. Huson, S. Nettles, and T. Warnow, "Disk-covering, a fast-converging method for phylogenetic tree reconstruction," *Journal of Computational Biology*, vol. 6, pp. 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)*. AAAI Press, 1999, pp. 118–129.

[9] 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)*, ser. Bioinformatics, vol. 17. Oxford Univeristy Press, 2001, pp. S190–S198.

[10] P. Goloboff, "Analyzing large data sets in reasonable times: solutions for composite optima," *Cladistics*, vol. 15, pp. 415–428, 1999.

[11] T. L. Williams and M. L. Smith, "The role of diverse populations in phylogenetic analysis," in *Genetic and Evolutionary Computation Conference (GECCO-2006)*, 2006, to appear.

[12] D. Maddison, "The discovery and importance of multiple island of most parsimonous trees," *Syst. Bio.*, vol. 42, no. 2, pp. 200–210, 1991.

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

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

[15] M. J. Brauer, M. T. Holder, L. A. Pries, D. J. Zwickl, P. O. Lewis, and D. M. Hillis, "Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference," *Mol. Biol. Evol.*, vol. 19, no. 10, pp. 1717–1726, 2002.

[16] C. B. Congdon, "Gaphyl: An evolutionary algorithms approach for the study of natural evolution," in *Genetic and Evolutionary Computation Conference (GECCO-2002)*, New York, NY, July 2002.

[17] A. G. A. for Maximum-Likelihood Phylogeny Inference Using Nucleotide Sequence Data, "The metapopulation genetic algorithm: An efficient solution for the problem of large phylogeny estimation," *Mol. Biol. Evol.*, vol. 15, no. 3, pp. 277–283, 1998.

[18] A. R. Lemmon and M. C. Milinkovitch, "The metapopulation genetic algorithm: An efficient solution for the problem of large phylogeny estimation," *PNAS*, vol. 99, no. 16, pp. 10 516–10 521, 2002.

[19] K. P. Johnson, "Taxon sampling and the phylogenetic position of passeriformes: Evidence from 916 avian cytochrome b sequences," *Systematic Biology*, vol. 50, no. 1, pp. 128–136, 2001.

[20] Scientific Computing Associates, Inc., "TCP Linda," Internet Website, last accessed, July 2005, sCAI's TCP Linda URL: http://www.lindaspaces.com/products/linda.html.

[21] D. Gelernter, "Generative communication in Linda," *ACM Transactions on Programming Languages and Systems*, vol. 7, no. 1, Jan. 1985.