

# Cooperative Rec-I-DCM3: A Population-Based Approach for Reconstructing Phylogenies

Tiffani L. Williams  
Department of Computer Science  
Texas A&M University  
tlw@cs.tamu.edu

Marc L. Smith  
Department of Computer Science  
Colby College  
mlsmith@colby.edu

**Abstract**—In this paper, we study the use of cooperation as a technique for designing faster algorithms for reconstructing phylogenetic trees. Our focus is on the use of cooperation to reconstruct trees based on maximum parsimony. Our baseline algorithm is Rec-I-DCM3, the best-performing MP algorithm known-to-date. Our results demonstrate that cooperation does improve the performance of the baseline algorithm by at least an order of magnitude in terms of running time. The use of cooperation also achieved new results in terms of the best-known scores, which the non-cooperative algorithm was unable to find.

## I. INTRODUCTION

A phylogeny is an evolutionary history of a set of organisms. Typically, it is represented by a tree, where the leaf nodes are modern-day organisms and the edges represent the relationships between the organisms. Phylogenetic analysis is used in all branches of biology with applications ranging from studies on the origin of human populations to investigations of the transmission patterns of HIV [15], and beyond, with a variety of uses in drug discovery, forensics, and security [1]. However, the ultimate goal of phylogenetics is inferring the “Tree of Life”, an evolutionary history of all-known organisms (or taxa). Current estimates of the size of the “Tree of Life” range from the millions to hundreds of million of taxa. Thus, new fast and accurate heuristics are desperately needed in order to have any chance of knowing the evolutionary relationships that exist between all organisms.

Yet, the accurate estimation of evolutionary trees is a very challenging computational problem. For  $n$  taxa, there are  $(2n - 5)!!$  possible phylogenies. Hence, there are over 13 billion possible trees for 13 taxa. Given that it is impossible to know the true evolutionary history of a set of organisms, the accuracy of a phylogenetic tree is often approximated by the use of NP-hard optimization criteria such as maximum parsimony (MP) or maximum likelihood (ML). By far, MP is the most commonly used criteria for reconstruction evolutionary trees [23]. It is based on Occam’s Razor, which states that the simplest explanation is the best. Hence, the tree that explains the

data with the fewest evolutionary events (i.e., mutations) is the one that is preferred under MP.

Currently, the best-performing MP heuristic is Recursive-Iterative DCM3 (Rec-I-DCM3) [22], [20]. Experimental results have shown that it outperforms its competitors by at least an order of magnitude on datasets ranging from 500 to 14,000 organisms. Although Rec-I-DCM3 is a good overall performer, each iteration of the search is guided by a single tree. However, could performance be improved if a population of trees is utilized instead? Good performance from a search algorithm is due to a balance of *exploration* (the generation of new individuals in untested regions of the search space) and *exploitation* (the concentration of the search in the vicinity of known good solutions). It is unclear if both of these objectives can be met by a local search heuristic that manipulates a single individual.

Memetic algorithms [16] combine a population-based global search and an individual local search. Unlike genetic algorithms, memetic algorithms exploit problem-domain knowledge through the local search algorithm. In this paper, we describe a new memetic-based technique called Cooperative Rec-I-DCM3, which maintains a population of Rec-I-DCM3 trees to search the tree landscape. Our algorithm is cooperative in the sense that the individuals in the population share pieces (i.e., subtrees) of themselves to create new individuals. Moreover, the presence of diverse solutions in the population represent a collective effort in finding accurate trees quickly.

We study the performance of our cooperative algorithm on two large biological datasets consisting of 921 and 2000 sequences. Our experimental results demonstrate that our cooperative approach outperforms Rec-I-DCM3 by at least an order of magnitude. Moreover, our algorithms establish new best-known scores on both of the datasets studied here.

## II. MAXIMUM PARSIMONY

The MP optimization problem seeks the tree with the minimum length, which is the smallest number of point mutations for the data. 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  $H(a, b)$ . (The Hamming distance between two strings of the same length is the number of positions in which they differ.) Let  $T$  be a tree whose nodes are labeled by sequences of length  $k$  over  $\Sigma$ , and let  $H(e)$  denote the Hamming distance of the sequences at each endpoint of  $e$ . Then 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 [6], but the problem of assigning sequences to internal nodes of a fixed leaf-labeled tree is polynomial [4]. Heuristics for MP use polynomial time solutions for scoring individual trees, along with techniques for moving through tree space, in order to find improved MP scores.

### A. 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 to find trees with a better score. One popular local search technique is based on using *Tree-Bisection and Reconnection (TBR)* [14] moves to locate better scoring trees. Figure 1 shows an example of how a tree is rearranged under TBR. In a TBR move, 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 algorithm [18] combines TBR (Tree- Bisection and Reconnection) hill-climbing with an interesting strategy to move out of local optima. Parsimony ratchet is an iterative algorithm, which begins from a “starting tree” created by the phylogenetic method of choice. Afterwards, a ratchet iteration consists of two searches. The first search is based on weighted MP, where the original dataset is randomly re-weighted so that the cost of a mutation is  $c$  instead of 1. TBR hill-climbing is used to find the best tree under the perturbed dataset. Once the first search gets stuck in a local optimum, the second search begins. The original state of the dataset is restored, and the search continues from the best tree found during the previous search. Once

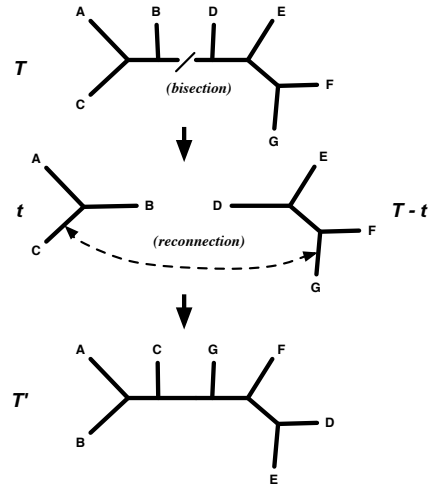


Fig. 1. Tree-Bisection and Reconnection. Here,  $T$  is split into two subtrees  $t$  and  $T - t$  at the internal edge labeled bisection. The two subtrees are then attached to the internal edges pointed to by the reconnection line.  $T'$  represents the resulting tree.

a local optimum is reached during this search, a ratchet iteration is complete.

### B. Disk-Covering Methods

Disk-Covering Methods (DCMs) [10], [11], [17], [21], [22] are an interesting approach to the reconstruction of phylogenetic trees. DCMs are typically incorporated into other existing phylogeny reconstruction techniques to improve performance. Collectively, DCMs are an example of divide-and-conquer algorithms that consist of four main stages, which we describe now, briefly.

First, divide the given tree into overlapping sets of leaf nodes (taxa). By overlapping, we mean sets of taxa with nonempty intersections. Second, solve the subproblems. This means, apply the phylogeny reconstruction technique at hand to solve each of the overlapping sets of taxa that resulted from the first stage. Each such solution is a tree whose leaf nodes correspond to one of the sets of taxa from stage one. Third, merge the solutions back together, recombining each of the solved subtrees back into a single tree. The hope is, the resulting tree at this stage is in some way better than the tree we began with during the first stage. Chances are, however, the resulting tree is also multifurcating. This presents a problem, because many existing phylogeny reconstruction techniques require bifurcating trees. Therefore, the fourth stage of a DCM consists of refining the tree from stage three so that it is a bifurcation.

While all DCMs consist of the four main stages just described, variations result from different strategies

during the first stage. We can strategize how to decompose subsets of taxa based on existing inter-taxa relationship measures, such as pairwise distances; or even leverage information from subtrees of the given tree. Regardless of the method of decomposition (which has a significant effect on overall performance of the DCM), the remaining three stages all work the same way. Thus, we have the family of disk-covering methods.

### III. REC-I-DCM3

Rec-I-DCM3 [22], [20] is the most recent addition to the DCM family. It was designed to avoid producing large subsets, as other DCM methods are prone to do. Subproblems produced from a very large dataset remain too large for immediate solution by a base method. Hence, the recursive application of the decomposition produces smaller and smaller subproblems until every subproblem is small enough to be solved directly. Once the dataset is decomposed into overlapping subsets, subtrees are constructed for each subset using the chosen base method, and then combined using the Strict Consensus Merger [10], [11] to produce a tree on the combined dataset.

### IV. COOPERATIVE REC-I-DCM3

Cooperative Rec-I-DCM3 maintains a population of solutions to balance exploitation and exploration during a search. Algorithm 1 presents the complete details of our algorithm. The algorithm starts by creating a collection of Greedy-MP trees, which are then subjected to TBR hill-climbing. To construct a Greedy-MP tree, we first randomize the ordering of the sequences in the dataset. Afterwards, the first three taxa are used to create an unrooted binary tree,  $T$ . The fourth taxon is added to the internal edge of  $T$  that results in the best MP score. This process is repeated until all taxa have been added to the tree.

The remainder of the algorithm proceeds iteratively, where each iteration consists of local search, selection, and recombination. The local search stage consists of applying one iteration of Rec-I-DCM3 to each of the trees in the population. Next, the trees are sorted based on their MP scores and placed into either set  $A$ ,  $B$ , or  $C$ . Trees in  $A \cup B$  can be considered to comprise highly fit individuals. These trees are then recombined with trees in  $C$  using strict consensus (see Figure 2). Since the consensus tree is typically multifurcating, the resulting consensus tree is randomly refined (see Figure 3) and TBR hill-climbing is applied. The above steps are repeated until the specified number of iterations are complete.

---

### Algorithm 1 Cooperative Rec-I-DCM3

---

```

1: Initialize  $pop$  with  $popsize$  Greedy-MP + TBR trees
2: repeat
3:   for  $i = 0$  to  $|p| - 1$  do
4:      $pop_i \leftarrow \text{Rec-I-DCM3}(pop_i)$ 
5:   end for
6:   {Divide  $pop$  into sets  $A, B$ , and  $C$ }
7:    $sortedPop \leftarrow \text{Sort}(pop)$ 
8:    $best \leftarrow \text{Min}(best, sortedPop_0)$ 
9:    $A \leftarrow \text{Copy}(best, nBest)$  { $nBest$  copies of  $best$ }
10:   $B \leftarrow \{sortedPop_i : 0 \leq i < nTop\}$ 
11:  {Select  $k$  trees from  $sortedPop - B$  for set  $C$ }
12:   $k \leftarrow |pop| - |A \cup B|$ 
13:   $C \leftarrow \text{SelectTrees}(sortedPop - B, k)$ 
14:  {Recombine  $A \cup B$  with  $C$  to create new trees}
15:  for  $i = 0$  to  $|C| - 1$  do
16:     $r \leftarrow \text{RandNum}(0, 1)$ 
17:    if  $r \leq recombPct$  then
18:       $tree1 \leftarrow \text{SelectTrees}(A \cup B, 1)$ 
19:       $tree2 \leftarrow C_i$ 
20:       $newTree \leftarrow \text{StrictConsensus}(tree1, tree2)$ 
21:      Refine  $newTree$  into a binary tree
22:      Apply TBR to  $newTree$ 
23:       $C_i \leftarrow newTree$ 
24:    end if
25:  end for
26:   $pop \leftarrow A \cup B \cup C$ 
27: until Required number of iterations

```

---

Although Cooperative Rec-I-DCM3 can be implemented on a sequential platform, our implementation takes advantage of the natural parallelism inherent in the algorithm. During the local search phase, each individual in the population is assigned to a processor and Rec-I-DCM3 is applied to it. However, the selection and recombination stages are executed on the same processor.

## V. EXPERIMENTAL METHODOLOGY

### A. Datasets

Our experiments compared the performance of the algorithms on two biological datasets containing 921 and 2000 sequences, respectively. Below, we provide the details of each dataset along with their best-known score under maximum parsimony since the optimal score is not known.

- 1) A set of 921 aligned Avian Cytochrome  $b$  DNA sequences from Kevin Johnson who published it in [12]. Roshan found a best score of 40496 on this dataset [20]. Our Cooperative Rec-I-DCM3 algorithm found a new best score of 40494.

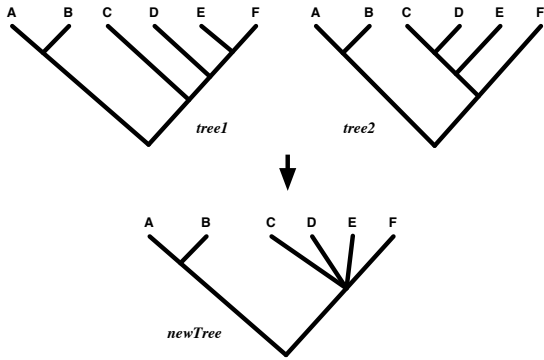


Fig. 2. Strict-consensus tree. A strict consensus preserves only those subtrees in common between the input trees. The subtree with leaves *A* and *B* is the only common subtree between *tree1* and *tree2*. However, the trees disagree on the placement of *C*, *D*, *E*, and *F*, which leads to the multifurcating node shown in *newTree*.

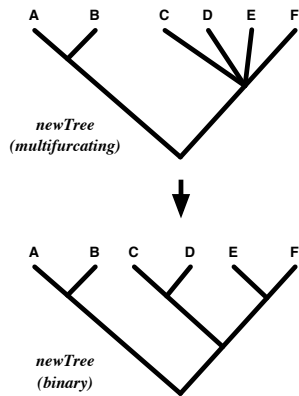


Fig. 3. Tree refinement. *newTree* has one multifurcating node of degree 4. One possible binary refinement of this node is shown by *newTree*.

- 2) A set of 2000 aligned Eukaryotic sRNA sequences (1251) obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin. Roshan found a best score of 74536 on this dataset [20]. Our runs of both Rec-I-DCM3 and Cooperative Rec-I-DCM3 established a new best score of 74534.

### B. Experiments

All experiments consisted of five runs of Rec-I-DCM3 and Cooperative Rec-I-DCM3. We ran Rec-I-DCM3 with the recommended default settings. Hence, the maximum subproblem sizes were set to 50% (461 taxa) and 25% (500 taxa) on Datasets 1 and 2, respectively. For Cooperative Rec-I-DCM3, we used the same settings for the local search phase of the algorithm.

Both Rec-I-DCM3 and Cooperative Rec-I-DCM3 were given sufficient time to find the best-known score. Hence, Rec-I-DCM3 ran for 1000 iterations, and its cooperative counterpart ran for 100 iterations with population sizes of 2, 4, 6, and 8 trees.

### C. Performance criteria

Most studies that compare the performance of MP algorithms use the MP score as an approximation of the goodness of the resulting tree. Better MP scores are thought to reflect improved accuracy of the inferred tree. We are interested in the relationship of the best scores found during a search and the best-known score for the dataset. In our experiments, we compute the best score,  $s_i$ , found by iteration  $i$ . We compute the number of steps,  $k$ , that  $s_i$  is from the best-known score,  $b$ . In other words,  $k = s_i - b$ .

### D. Implementation

We used TCP Linda [24], an implementation of Gelernter's Linda [7] model of concurrency, to implement our cooperative algorithm. 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 was used as the base method for Rec-I-DCM3, and we used TNT's implementation of TBR. We used PAUP\*'s implementation of strict consensus.

### E. Platforms

Our experiments were performed on two high-performance computing clusters: an Apple Workgroup Cluster for Bioinformatics and a Linux Beowulf cluster. Both clusters are similarly configured, each consisting of four, 64-bit, dual-processor nodes (eight total CPUs) with gigabit-switched interconnects.

However, the underlying hardware of the clusters is quite different. The Apple Workgroup 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). The Linux Beowulf cluster consists of four nodes; each node contains two, 2 GHz Intel Xeon processors. Each processor contains 512 KB of L2 cache, but only a 400 MHz front-side bus; the two processors on each node share 2 GB of DDR 266 MHz SDRAM (8 GB total RAM across the cluster).

## VI. EXPERIMENTAL RESULTS

The objective of our experimental analysis is to compare the performance of our cooperative algorithm with its non-cooperative counterpart. In our tables and figures, the  $popsiz = 1$  curve refers to Rec-I-DCM3, and larger population sizes refer to Cooperative Rec-I-DCM3. Of particular interest are the following questions.

- 1) What is the cost of maintaining a population of solutions?
- 2) Can a cooperative algorithm outperform its non-cooperative counterpart?
- 3) What role does recombination play in the performance of a cooperative algorithm?

We consider each of the above questions, in turn, in the following subsections.

### A. Running Time

Table I provides the running time for Rec-I-DCM3 and Cooperative Rec-I-DCM3. For a population size of 2, the running times are essentially the same. As there is only one tree available for recombination, the chance of recombination is very small. Hence, the low overhead. However, as the population size increases so does the overhead of using the cooperative algorithm. We should also note that the heterogeneity of our experimental platforms did not affect the performance. Thus, the time per iteration of the algorithms were quite similar regardless of which platform was used.

For a population of size eight, the overhead of using Cooperative Rec-I-DCM3 is approximately twice that of Rec-I-DCM3. However, this is a result of not being able to assign the eight individuals to a separate processor. Our implementation uses additional processors to manage the termination of the cooperative computation and to create a new population. Thus, the maximum number of Rec-I-DCM3 individuals we could allocate to a separate processor was six. For populations of size eight, some of the workers were responsible for processing more than one tree. We should also point out, as will be shown in later sections, that given the overhead of using a population size of eight, it still clearly outperforms Rec-I-DCM3 – even when Rec-I-DCM3 is allowed to execute longer than Cooperative Rec-I-DCM3.

### B. Rec-I-DCM3 versus Cooperative Rec-I-DCM3

Figure 4 shows the average performance of Rec-I-DCM3 and Cooperative Rec-I-DCM3 on Dataset 1 for 100 iterations. Here, the plots show that after 40 iterations our cooperative algorithm outperforms Rec-I-DCM3 for all population sizes. The plot also shows

$popsiz$	Dataset 1		Dataset 2	
	iteration (secs)	total (hrs)	iteration (secs)	total (hrs)
1	54.00	15.00†	157.64	43.74†
2	55.08	1.53	157.68	4.38
4	58.68	1.63	184.32	5.12
6	72.00	2.00	219.96	6.11
8	114.12	3.17	335.16	9.31

TABLE I

AVERAGE RUNNING TIMES FOR REC-I-DCM3 ( $popsiz=1$ ) AND COOPERATIVE REC-I-DCM3. † IS THE TIME TO RUN REC-I-DCM3 FOR 1000 ITERATIONS.

that as the number of trees in the population increases, so does the performance of the cooperative algorithm, with a population of size eight being the best performer. After 100 iterations, the cooperative algorithm with eight individuals is within 7 steps of the best-known score compared to 13 steps for Rec-I-DCM3.

Given that Cooperative Rec-I-DCM3 clearly outperforms its competitor over 100 iterations, we allowed Rec-I-DCM3 to run for a total of 1000 iterations. Figure 5 shows the results. With 1000 iterations, Rec-I-DCM3 surpasses Cooperative Rec-I-DCM3 with a population size of two. However, it cannot reach the performance of Cooperative Rec-I-DCM3 with larger population sizes. After 1000 iterations, Rec-I-DCM3 is 9 steps from the best-known score, compared to 7 steps for the cooperative algorithm with a population size of eight.

Figure 6 shows the performance of the algorithms on Dataset 2. Cooperative Rec-I-DCM3 outperforms Rec-I-DCM3 at every iteration. By iteration 60, all five runs of the Cooperative Rec-I-DCM3 with eight individuals have found the best-known score. Again, we allow Rec-I-DCM3 to run for 1000 iterations to see if it can surpass the performance of the cooperative algorithm at 100 iterations. Rec-I-DCM3 does surpass the performance of the cooperative algorithm with a population of size 2. However, it cannot reach the performance of Cooperative Rec-I-DCM3 with larger population sizes.

Table II shows the number of Rec-I-DCM3 and Cooperative Rec-I-DCM3 runs that find the best-known score. For Dataset 1, we see that Cooperative Rec-I-DCM3 with a population size of 6 is able to find the best-known score. On the other hand, for dataset 2, all algorithms were able to find the best-known score, except for Cooperative Rec-I-DCM3 with a population of 2. Cooperative Rec-I-DCM3 with a population size of 8 was able to find the best-known score over all 5 runs.

### C. Improvement under recombination

Figures 8 and 9 show the performance of the recombination operator on Datasets 1 and 2, respectively. On

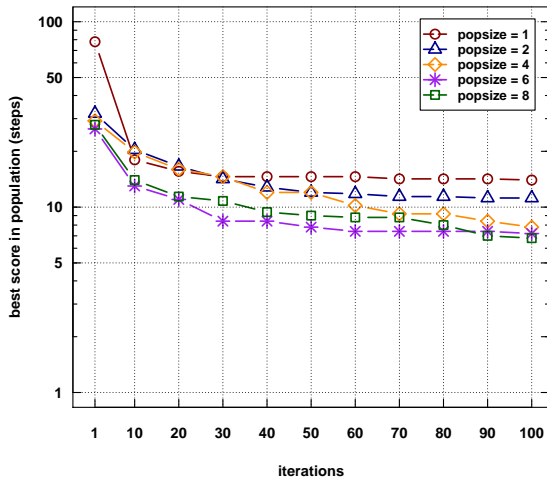


Fig. 4. The performance of Rec-I-DCM3 and Cooperative Rec-I-DCM3 on Dataset 1 (921 taxa) over 100 iterations. The  $popsize = 1$  curve corresponds to Rec-I-DCM3, whereas the other curves correspond to Cooperative Rec-I-DCM3 using various population sizes. The y-axis is on a log-scale.

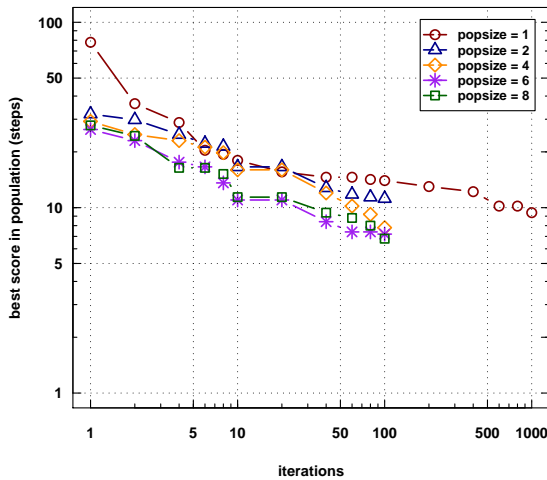


Fig. 5. Extending the runs of Rec-I-DCM3 for 1000 iterations on Dataset 1 (921 taxa). Cooperative Rec-I-DCM3 ran for 100 iterations. The  $popsize = 1$  curve corresponds to Rec-I-DCM3, whereas the other curves correspond to Cooperative Rec-I-DCM3 using various population sizes. Both axes are on a log-scale.

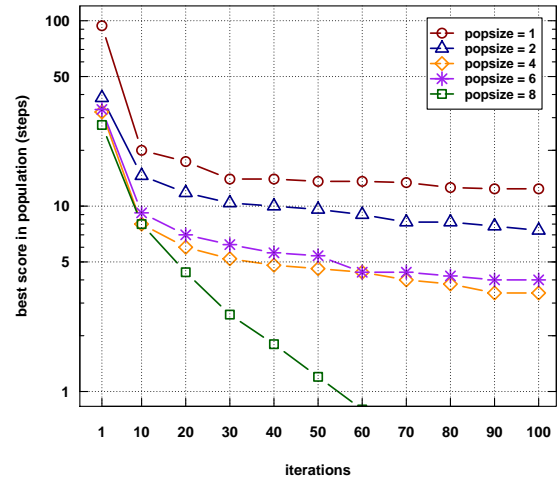


Fig. 6. The performance of Rec-I-DCM3 and Cooperative Rec-I-DCM3 on Dataset 2 (2000 taxa) for 100 iterations. The  $popsize = 1$  curve corresponds to Rec-I-DCM3, whereas the other curves correspond to Cooperative Rec-I-DCM3 using various population sizes. Since the y-axis is plotted on a log scale, scores that are 0 steps from the best-known score are undefined. Hence, the  $popsize = 8$  curve is undefined after iteration 60 since it reaches the best-known score.

$popsize$	Dataset 1	Dataset 2
1	0	1
2	0	0
4	0	1
6	1	2
8	0	5

TABLE II

NUMBER OF RUNS FINDING THE BEST-KNOWN SCORE  
REC-I-DCM3 ( $popsize=1$ ) AND COOPERATIVE REC-I-DCM3.

Dataset 1, recombination helps during the early stages of the search, but afterwards, it is more advantageous not to use recombination. However, on Dataset 2, with a population size of 4 and 8, recombination improves the performance of the algorithm. Yet, recombination seems to hurt the algorithm for population sizes of 2 and 6.

## VII. RELATED WORK AND DISCUSSION

The results from our experiments show that our cooperative algorithm results in improved performance when compared to its non-cooperative counterpart. The success of our approach can be attributed to using a high-performance local search algorithm such as Rec-I-DCM3 as the basis for constructing a population of diverse

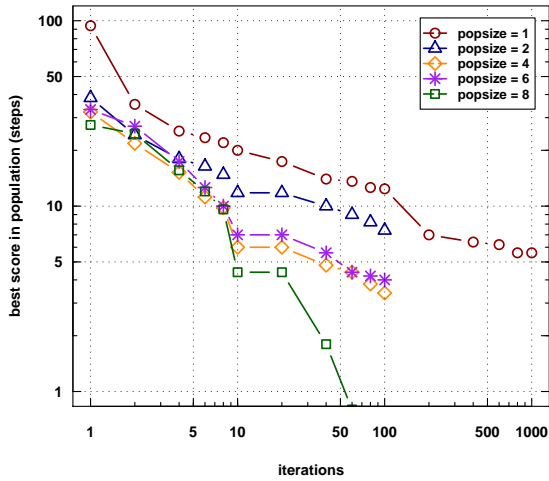


Fig. 7. Extending the runs of Rec-I-DCM3 for 1000 iterations on Dataset 2 (2000 taxa). Cooperative Rec-I-DCM3 ran for 100 iterations. The *popsize* = 1 curve corresponds to Rec-I-DCM3, whereas the other curves correspond to Cooperative Rec-I-DCM3 using various population sizes. Both axes are plotted on a log-scale, and are undefined at 0. Hence, the *popsize* = 8 curve is undefined after iteration 60 since it reaches the best-known score.

individuals to guide the search. Moreover, our results demonstrate that large population sizes are not needed in order to see improved performance. Even with a population consisting of two individuals, our cooperative algorithm finds better scores much faster than Rec-I-DCM3. Since good performance can be achieved while using a very small population of individuals, the implementation of Cooperative Rec-I-DCM3 is quite feasible on sequential platforms.

Population-based approaches such as genetic algorithms have also been applied to maximum likelihood (ML) methods [2], [3], [5], [13] since ML is more computationally intensive than MP. The performance of these genetic algorithms are often not compared to the top performing ML algorithms, which include fastDNAML [19], MrBayes [9], and PHYML [8]. Thus, it is difficult to evaluate the performance of the algorithms when the top competitors are missing. Here, we explicitly compare our cooperative algorithm to Rec-I-DCM3, which is the best MP algorithm available. Moreover, Rec-I-DCM3 has been extensively tested against other MP approaches such as TNT and parsimony ratchet and has consistently outperformed them over a variety of different datasets.

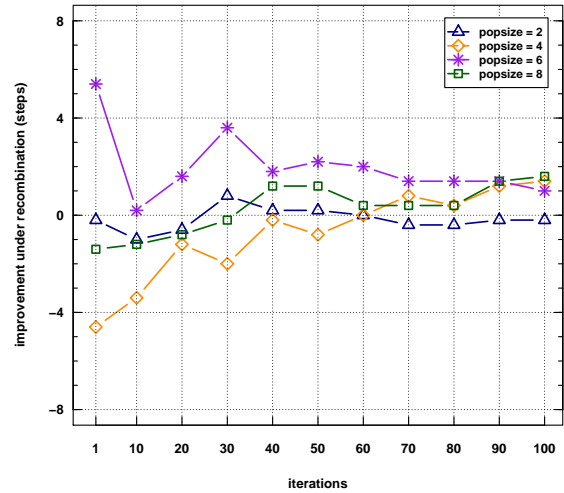


Fig. 8. Improvement gained by using recombination to create new trees on Dataset 1. The y-axis indicates the steps of improvement gained under Cooperative Rec-I-DCM3 by using a recombination operator. Thus, positive (negative) values indicate that recombination improves (worsens) the performance of the algorithm when compared to using Cooperative Rec-I-DCM3 without recombination.

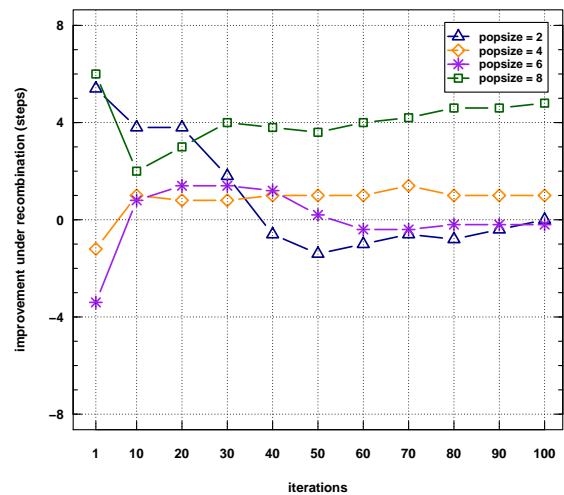


Fig. 9. Improvement gained by using recombination to create new trees on Dataset 2. The y-axis indicates the steps of improvement gained under Cooperative Rec-I-DCM3 by using a recombination operator. Thus, positive (negative) values indicate that recombination improves (worsens) the performance of the algorithm when compared to using Cooperative Rec-I-DCM3 without recombination.

## VIII. CONCLUSIONS

We have presented a new algorithm called Cooperative Rec-I-DCM3 that convincingly outperforms Rec-I-DCM3, the best performing MP heuristic to date, by at least an order of magnitude. Moreover, the algorithm improved upon the best-known scores for the datasets studied here. Our algorithm utilizes a population of trees, which cooperatively search through tree space by sharing pieces (i.e., subtrees) of themselves to create new individuals. The diversity of solutions in the population allows our algorithm to maintain a balance between two contradictory search objectives: exploitation and exploration.

Yet, the most exciting aspect of our study is the wide applicability of employing cooperative searches to reconstruct evolutionary trees. Here, we limited our study to improving the performance of Rec-I-DCM3. However, the performance of other MP approaches such as parsimony ratchet, TBR hill-climbing, or a heterogeneous mixture of methods should benefit from our cooperative search framework. We also plan to investigate other techniques for selecting and recombining individuals. Lastly, although maximum parsimony was the focus of this study, our approach should be applicable to maximum likelihood as well.

## IX. ACKNOWLEDGMENTS

Most of this work was done while Williams was at the Radcliffe Institute of Advanced Study, and Smith was on sabbatical leave from Colby College. The authors would also like to thank Usman Roshan for providing the code for Rec-I-DCM3 and the datasets to use for this study.

## REFERENCES

- [1] D. Bader, B. M. Moret, and L. Vawter. Industrial applications of high-performance computing for phylogeny reconstruction. In H. Siegel, editor, *Proceedings of SPIE Commercial Applications for High-Performance Computing*, volume 4528, pages 159–168, Denver, CO, Aug. 2001. SPIE.
- [2] 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.*, 19(10):1717–1726, 2002.
- [3] 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.
- [4] W. M. Fitch. Toward defining the course of evolution: minimal change for a specific tree topology. *Syst. Zool.*, 20:406–416, 1971.
- [5] 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.*, 15(3):277–283, 1998.
- [6] L. R. Foulds and R. L. Graham. The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.
- [7] D. Gelernter. Generative communication in linda. *ACM Transactions on Programming Languages and Systems*, 7(1), Jan. 1985.
- [8] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst. Biol.*, 52(5):696–704, 2003.
- [9] J. P. Huelsenbeck and F. Ronquist. MRBAYES: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17(8):754–755, 2001.
- [10] 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.
- [11] 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.
- [12] K. P. Johnson. Taxon sampling and the phylogenetic position of passeriformes: Evidence from 916 avian cytochrome b sequences. *Systematic Biology*, 50(1):128–136, 2001.
- [13] A. R. Lemmon and M. C. Milinkovitch. The metapopulation genetic algorithm: An efficient solution for the problem of large phylogeny estimation. *PNAS*, 99(16):10516–10521, 2002.
- [14] D. Maddison. The discovery and importance of multiple island of most parsimonious trees. *Syst. Bio.*, 42(2):200–210, 1991.
- [15] M. L. Metzker, D. P. Mindell, X.-M. Liu, R. G. Ptak, R. A. Gibbs, and D. M. Hillis. Molecular evidence of HIV-1 transmission in a criminal case. *PNAS*, 99(2):14292–14297, 2002.
- [16] P. A. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memtic algorithms. Technical report, Caltech, 1989.
- [17] 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 Univeristy Press, 2001.
- [18] K. C. Nixon. The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics*, 15:407–414, 1999.
- [19] G. Olsen, H. Matsuda, R. Hagstrom, and R. Overbeek. fastdnaml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Comput. Appl. Biosci.*, 10:41–48, 1994.
- [20] U. Roshan. *Algorithmic Techniques for Improving the Speed and Accuracy of Phylogenetic Methods*. PhD thesis, The University of Texas at Austin, May 2004.
- [21] U. Roshan, B. M. Moret, T. Warnow, and T. L. Williams. Performance of supertree methods on various dataset decompositions. In O. Bininda-Emonds, editor, *Phylogenetic Supertrees: Combining information to reveal the Tree of Life.*, pages 301–328. Kluwer Acad. Publ., Dordrecht, 2004.
- [22] 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)*, pages 98–109. IEEE Press, 2004.
- [23] M. Sanderson, B. Baldwin, G. Bharathan, C. Campbell, D. Ferguson, J. Porter, C. V. Dohlen, M. Wojciechowski, and M. Donoghue. The growth of phylogenetic information and the need for a phylogenetic database. *Systematic Biology*, 42:562–568, 1993.
- [24] Scientific Computing Associates, Inc. Tep linda. Internet Website, last accessed, July 2005. SCAI's TCP Linda URL: <http://www.lindaspaces.com/products/linda.html>.