Parallel Divide-and-Conquer Phylogeny Reconstruction by Maximum Likelihood*

Z. Du[1], A. Stamatakis[2], F. Lin[1], U. Roshan[3], L. Nakhleh[4]

[1] Bioinformatics Research Center, Nanyang Technological University, Nanyang Avenue, Singapore 639798
[2] Institute of Computer Science, Foundation for Research and Technology-Hellas, P.O. Box 1385, Heraklion, Crete, GR-71110 Greece
[3] College of Computing Sciences, Computer Sciences Department, New Jersey Institute of Technology, University Heights, Newark, NJ 07102
[4] Department of Computer Science, Rice University, 6100 Main St. MS 132, Houston, TX 77005

Abstract. Phylogenetic trees are important in biology since their applications range from determining protein function to understanding the evolution of species. Maximum Likelihood (ML) is a popular optimization criterion in phylogenetics. However, inference of phylogenies with ML is NP-hard. Recursive-Iterative-DCM3 (Rec-I-DCM3) is a divide-and-conquer framework that divides a dataset into smaller subsets (subproblems), applies an external *base method* to infer subtrees, merges the subtrees into a comprehensive tree, and then refines the global tree with an external *global method*. In this study we present a novel parallel implementation of Rec-I-DCM3 for inference of large trees with ML. Parallel-Rec-I-DCM3 uses RAxML as external base *and* global search method. We evaluate program performance on 6 large real-data alignments containing 500 up to 7.769 sequences. Our experiments show that P-Rec-I-DCM3 reduces inference times and improves final tree quality over sequential Rec-I-DCM3 and stand-alone RAxML.

1 Introduction

Phylogenetic trees describe the evolutionary relationship among a group of organisms and are important for answering many biological questions. Some applications of phylogenetic trees are multiple sequence alignment [5], protein structure prediction [19], and protein function prediction [12]. Several research groups such as CIPRES (www.phylo.org) and ATOL (tolweb.org) are working on novel heuristics for the reconstruction of very large phylogenies.

There exist two basic models for calculating phylogenetic trees based on DNA or Protein sequence data: distance-based and character-based methods. *Distance-based* methods construct a tree from a matrix of pairwise distances between the input sequences. Current popular distance-based methods such as

* Part of this work is funded by a Postdoc-fellowship granted by the German Academic Exchange Service (DAAD)

Neighbor Joining [18] (NJ) tolerate only a limited amount of error in the input; thus, they do not perform well under challenging conditions [15]. The two most popular character based optimization criteria are Maximum Parsimony (MP) [9] and Maximum Likelihood (ML) [6]. The optimal ML-tree is the tree topology which is most likely to have given rise to the observed data under a given statistical model of sequence evolution. This problem has always been known to be very hard in practice—the number of different tree topologies grows exponentially with the number of sequences n, e.g. for $n = 50$ organisms there already exist $2.84 * 10^{76}$ alternative topologies; a number almost as large as the number of atoms in the universe ($\approx 10^{80}$)—and was recerntly proven to be NP-hard [3].

For constructing the Tree of Life, i.e., the evolutionary tree on all species on Earth, efficient heuristics are required which allow for analysis of large and complex datasets in reasonable time, i.e. in the order of weeks instead of years. Heuristics that can rapidly solve ML are of enormous benefit to the biological community. In order to accelerate the analysis of large datastes with thousands of sequences via a divide and conquer approach the family of Disk Covering Methods (DCMs) [8, 17] has been introduced. DCMs are divide and conquer methods which decompose the dataset, computes subtrees on smaller subproblems using a *base method* and then merge the subtrees to yield a tree on the full dataset. To the best of our knowledge Recursive-Iterative-DCM3 (Rec-I-DCM3) [17] is the best-known DCM in conjunction with the MP and ML criteria. In combination with Rec-I-DCM3 we use RAxML [20] as base method and a re-implementation of parallel RAxML [21] as global method. RAxML is among the currently fastest, most accurate, as well as most memory-efficient ML heuristics on real biological datasets. The goal of the parallel MPI implementation is to exploit the computational power of PC clusters, i.e. to calculate better trees within the same amount of total execution time. P-Rec-I-DCM3 is available as open source software from the authors. It can be compiled and executed on any Linux PC cluster.

2 Related Work

The survey of related work is restrained to parallel/distributed implementations of maximum likelihood programs and to divide and conquer approaches. The phylogenetic navigator (PHYNAV [13]) which is based on a zoom-in/zoom-out approach represents an interesting alternative to Rec-I-DCM3. However, the program has a relatively high memory consumption (crashed on a 1.000-taxon tree with 1GB RAM) compared to RAxML. Furthermore, the global optimization method (fast Nearest Neighbor Interchange adapted from PHYML [7]) is not as efficient on real alignment data as RAxML [20]. Thus, it is not suited to handle large real-data alignments of more than 1.000 sequences.

Despite the fact that parallel implementations of ML programs are technically very solid, they significantly drag behind algorithmic development. This means that programs are parallelized that mostly do not represent the state-of-the-art algorithms any more. Therefore, they are likely to be out-competed by

the most recent sequential algorithms. For example the largest treec computed with parallel fastDNAml [22] which is based on the fastDNAml algorithm (1994) contains 150 taxa. The same holds for a technically very interesting JAVA-based distributed implementation of fastDNAml: DPRml [11]. Performance penalties are also caused by using JAVA due to unfavorable memory efficiency and speed of numerical calculations. Those language-dependent limitations will become more significant when trees comprising more than 417 taxa (currently largest tree with DPRml, personal communication) are computed with DPRml. M.J. Brauer et al. [2] have implemented a parallel genetic tree-search algorithm which has been used to compute trees of up to approximately 3.000 taxa with the main limitation being also memory consumption (personal communication).

Finally, there exists the previous parallel implementation of RAxML which has been used to compute one of the largest ML-based phylogenies to date containing 10.000 organisms [21]. Due to the high memory efficiency of RAxML (which the program inherited from fastDNAml) and the good performance on large real-world data it appears to be best-suited for use with Rec-I-DCM3. The goal of P-Rec-I-DCM3(RAxML) consists in a further acceleration of the algorithm.

3 Our new method: Parallel Recursive-Iterative-DCM3

3.1 Parallelizing Recursive-Iterative-DCM3

Rec-I-DCM3 is the latest in the family of Disk Covering Methods (DCMs) which were originally introduced by Warnow *et. al.* [8]. Rec-I-DCM3 [17] was designed to improve the performance of MP and ML heuristics. The method applies an external *base method* (any MP or ML heuristic), to smaller subsets of the full datasets (which we call subproblems). The division into subproblems is executed recursively until all subproblems contain less taxa than the user-specified maximum subproblem size m. The subproblems which are smaller in size and evolutionary diameter [17], are easier and faster to analyze. Rec-I-DCM3 has been shown to significantly improve upon heuristics for MP [17]. In this paper we initially show that sequential Rec-I-DCM3 with RAxML as external method can improve upon stand-alone RAxML for solving ML. Parallel Rec-I-DCM3 is a modification of the original Rec-I-DCM3 method. The P-Rec-I-DCM3 algorithm is outlined below (differences from the sequential version are highlighted in bold letters).

Outline of Parallel Rec-I-DCM3

- *Input:*
  - Input alignment S, #iterations n, base heuristic b, parallel global search method g, starting tree T, maximum subproblem size m
- *Output:* Phylogenetic tree leaf-labeled by S.
- *Algorithm:* For each iteration do
  - **Set T' = Parallel-Recursive-DCM3**(S, m, b, T)**.**

- **Apply the parallel global search method g starting from T' until we reach a local optimum.** Let T'' be the resulting local optimum.
- Set $T = T''$.

The Parallel-Recursive-DCM3 routine performs the work of dividing the dataset into smaller subsets, solving the subproblems in parallel (using the base method) with a master-worker scheme, and then merging the subtrees into the full tree. However, the sizes of individual subproblems vary significantly and the inference time per subproblem is not known a priori and difficult to estimate. This can lead to significant load imbalance (see Section 4.3). A distinct method of subtree-decomposition which yields subproblems of equal size for better load-balance does not appear promising (unpublished implementation in RAxML). This is due to the fact that Rec-I-DCM3 constructs subproblems intelligently with regard to closely-related taxa based on the information of the guide tree.

The global search method further improves the accuracy of the Recursive-DCM3 tree and can also find optimal global configurations that were not found by Recursive-DCM3, which only operates on smaller—local—subsets. To effectively parallelize Rec-I-DCM3 one has to parallelize the Recursive-DCM3 and the global search method. A previous implementation of parallel Rec-I-DCM3 for Maximum Parsimony [4] only parallelizes Recursive-DCM3 and not the global method. Our unpublished studies show that only parallelizing Recursive-DCM3 does not significantly improve performance over the serial Rec-I-DCM3. Our Parallel Rec-I-DCM3, however, also parallelizes the global search method, a key and computationally expensive component of Rec-I-DCM3. Note, that due to the complexity of the problem and the ML criterion it is not possible to avoid global optimizations of the tree altogether. All divide and conquer approaches for ML to date execute global optimizations at some point (see Section 2). For this study we use sequential RAxML as the base method and a re-implementation of parallel RAxML (developed in this paper) as the global search method.

3.2 Parallelizing RAxML

In this Section we provide a brief outline of the RAxML algorithm, which is required to understand the difficulties which arise with the parallelization.

Provided a comprehensive starting tree (for details see [20]), the likelihood of the topology is improved by subsequent application of topological alterations. To evaluate and select alternative topologies RAxML uses a mechanism called *lazy subtree rearrangements* [20]. This mechanism initially performs a rapid pre-scoring of a large number of topologies. After the pre-scoring step a few (20) of the best pre-scored topologies are analyzed more thoroughly. To the best of our knowledge, RAxML is currently among the fastest *and* most accurate programs on real alignment data due to this ability to quickly pre-score a large number of alternative tree topologies.

As outlined in the Figure available on-line at [1] the optimization process can be classified into two main computational phases:

Difficult parallelization: The *Initial Optimization Phase (IOP)* where the likelihood increases steeply and many improved pre-scored topologies are found during a single iteration of RAxML.

Straight-forward Parallelization: The *Final Optimization Phase (FOP)* where the likelihood improves asymptotically and practically all improved topologies are obtained by thoroughly optimizing the 20 best pre-scored trees.

The difficulties regarding the parallelization of RAxML are mainly due to hard-to-resolve dependencies caused by the detection of many improved trees during the IOP. Moreover, the *fast* version of the hill-climbing algorithm of RAxML which is used for global optimization with Rec-I-DCM3 further intensifies this problem, since it terminates after the IOP. During one iteration of RAxML all n subtrees of the candidate topology are subsequently removed and re-inserted into neighboring branches. The dependency occurs when the lazy rearrangement of a subtree i yields a topology with a better likelihood than the candidate topology even though it is only pre-scored. In this case the improved topology is kept and rearrangement of subtree $i+1$ is performed on the new topology. Especially, during the IOP improved pre-scored topologies are frequently encountered in the course of one iteration, i.e. n lazy subtree rearrangements. Since the lazy rearrangement of *one* single subtree is fast, a coarse-grained MPI-parallelization can only be based on assigning the rearrangement of distinct subtrees within the current candidate tree simultaneously to the workers. This represents a non-deterministic solution to the potential dependencies between rearrangements of subtrees i and $i+1$. This means that when two workers w_0 and w_1 simultaneously rearrange subtrees i and $i+1$ within the currently best tree and the rearrangement of subtree i yields a better tree, worker w_1 will miss this improvement since it is still working on the old tree. It is this frequently occurring dependency during the IOP between steps $i \rightarrow i+1$ ($i = 1...n$, $n =$ number of subtrees) which leads to parallel performance penalties. Moreover, this causes a non-deterministic behavior since the parallel program traverses another path in search space each time and might yield better or worse final tree topologies compared to the sequential program. The scalability for smaller number of processors is better since every worker misses less improved trees.

The aforementioned problems have a significant impact on the IOP only, since the FOP can be parallelized more efficiently. Furthermore, due to the significantly larger proportion of computational time required by the FOP the parallel performance of the slow hill-climbing version of RAxML is substantially better (see [21] for details). The necessity to parallelize and improve performance of RAxML fast hill-climbing has only been recognized within the context of using RAxML in conjunction with Rec-I-DCM3 and is an issue of future work.

3.3 Parallel Recursive-Iterative-DCM3

In this section we summarize how Rec-I-DCM3 and RAxML are integrated into one single parallel program. A figure with the overall program flow is available online at [1]. The parallelization is based on a simple master-worker architecture. The master initially reads the alignment file and initial guide tree. Thereafter, it

performs the DCM-based division of the problem into smaller subproblems and stores the merging order which is required to correctly execute the merging step. All individual subproblems are then distributed to the workers which locally solve them with sequential RAxML and then return the respective subtrees to the master. Once all subproblems have been solved the master merges them according to the stored merging order into the *new guide tree*. This guide tree is then further optimized by parallel RAxML. In this case the master distributes the IDs of the subtrees (simple integers) which have to be rearranged along with the currently best tree (only if it has improved) to the worker processes. The worker rearranges the specified subtree within the currently best tree and returns the tree topology (only if it has a better likelihood) along with a new work request. This process continues until no subtree rearrangement can further improve upon the tree. Finally, the master verifies if the specified amount of P-Rec-I-DCM3 iterations has already been executed and terminates the program. Otherwise, it will initiate a new round of subproblem decomposition, subproblem inference, subtree merging, and global optimization. The time required for subproblem decomposition and subtree merging is negligible compared to ML inference times.

4 Results

Test Datasets: We used a large variety of biological datasets ranging in size and type (DNA or RNA). The datasets have been downloaded from public databases or have been obtained from researchers who have manually inspected and verified the alignments: **Dataset1:** 500 *rbcL* DNA sequences (1398 sites) [16], **Dataset2:** 2,560 *rbcL* DNA sequences (1,232 sites) [10], **Dataset3:** 4,114 16s ribosomal Actinobacteria RNA sequences (1,263 sites) [14], **Dataset4:** 6,281 small subunit ribosomal Eukaryotes RNA sequences (1,661 sites) [24], **Dataset5:** 6,458 16s ribosomal Firmicutes (bacteria) RNA sequences (1,352 sites) [14], **Dataset6:** 7,769 ribosomal RNA sequences (851 sites) from three phylogenetic domains, plus organelles (mitochondria and chloroplast), obtained from the Gutell Lab at the Institute for Cellular and Molecular Biology, The University of Texas at Austin.

Test Platform: P-Rec-I-DCM3 and RAxML are implemented in C and use MPI. As test platform we used a cluster with 16 customized Alpha Server ES45 compute nodes; each node is equipped with 4 Alpha-EV68 1GHz processors and has 8 GB/s memory bandwidth and an interconnect PCI adapter with over 280 MB/s of sustained bandwidth. The central component of the cluster consists of a Quadrics 128-port interconnect switch chassis which delivers up to 500 MB/s per node, with 32 GB/s of cross-section bandwidth and MPI application latencies of less than 5 microseconds.

4.1 Sequential Performance

In the first set of experiments we examine the sequential performance of standalone RAxML over Rec-I-DCM3(RAxML). The respective maximum subset sizes

of Rec-I-DCM3 are adapted to the size of each dataset: Dataset1 (max. subset size: 100 taxa), Datasets2 (125 taxa), Dataset3 to Dataset6 (500 taxa). In our experiments both methods start optimizations on the *same* starting tree. Due to the relatively long execution times we only executed one Rec-I-DCM3 iteration per dataset. The run time of one Rec-I-DCM3 iteration was then used as inference time limit for RAxML. Table 1 provides the log likelihood values for RAxML and Rec-I-DCM3 after the same amount of execution time. Note that, the apparently small differences in final likelihood values *are significant* because those are logarithmic values and due to the requirements for high score accuracy in phylogenetics [23]. The experiments clearly show that Rec-I-DCM3 improves over stand-alone RAxML on *all* datasets (a more thorough performance study is in preparation).

Table 1. Rec-I-DCM3 versus RAxML log likelihood (LLH) values after the same amount of time

Dataset	Rec-I-DCM3 LLH	RAxML LLH	Dataset	Rec-I-DCM3 LLH	RAxML LLH
Dataset1	-99967	-99982	Dataset4	-1270920	-1271756
Dataset2	-355071	-355342	Dataset5	-901904	-902458
Dataset3	-383578	-383988	Dataset6	-541255	-541438

4.2 Parallel Performance

In our second set of experiments we assess the performance gains of P-Rec-I-DCM3 over the original sequential version. For each dataset we executed three individual runs with Rec-I-DCM3 and P-Rec-I-DCM3 on 4, 8, and 16 processors respectively. Each individual sequential and parallel run was executed ueing the *same* starting tree and the previously indicated subset sizes. In order to determine the speedup we measured the execution time of one sequential and parallel Rec-I-DCM3 iteration for each dataset/number of processors combination. The average sequential and parallel execution times per dataset and number of processors over three individual runs are available on-line at [1]. Due to the dependencies in parallel RAxML and the load imbalance the overall speedup and scalability of P-Rec-I-DCM3 are moderate. However, we consider the present work as proof-of-concept implementation to demonstrate the benefits from using P-Rec-I-DCM3 with RAxML and to analyze the technical and algorithmic challenges which arise with the parallelization. A separate analysis of the speedup values in Table 2 for the *base*, *global*, and *whole method* shows that the performance penalties originate mainly from parallel RAxML. Note however, that in order to improve the unsatisfying performance of parallel RAxML it executes a slightly more thorough search than the sequential global optimization with RAxML. Therefore, the comparison can not only be based on speedup values but must also consider the significantly better final likelihood values of P-Rec-I-DCM3. To demonstrate this P-Rec-I-DCM3 is granted the overall execution

time of one sequential Rec-I-DCM3 iteration (same response time). The final log likelihood values of Rec-I-DCM3 and P-Rec-I-DCM3 (on 16 processors) after the same amount of total execution time are listed in Table 3. Note that, the apparently small differences in final likelihood values *are significant*. Furthermore, the computational effort to attain those improvements is not negligible due to the asymptotic increase of the log likelihood in the *FOP* (see Section 3.2).

Table 2. Average base method, global method, and overall speedup values (over three runs) for P-Rec-I-DCM3 over Rec-I-DCM3 for one iteration of each method.

Procs	base	global	overall	procs	base	global	overall	procs	base	global	overall
Dataset1				Dataset2				Dataset3			
4	4	2.4	2.6	4	3	2.68	2.7	4	1.95	2.6	2.2
8	4.7	2.8	3.6	8	5.3	3.2	3.45	8	5.5	5	5.3
16	4.85	2.78	3.5	16	7	4.2	4.6	16	6.7	5.7	6.2
Dataset4				Dataset5				Dataset6			
4	2.9	2.3	2.6	4	2.3	2.7	2.5	4	3.2	1.95	2.2
8	4.2	4.9	4.6	8	4.8	4.4	4.7	8	4.8	2.5	3
16	8.3	5.3	6.3	16	7.6	5.1	5.8	16	5.4	2.8	3.3

Table 3. Average Log likelihood (LLH) scores of Rec-I-DCM3 (Sequential) and P-Rec-I-DCM3 (Parallel, on 16 processors) per dataset after the same amount of total execution time over three individual runs.

Dataset	Sequential LLH	Parallel LLH	Dataset	Sequential LLH	Parallel LLH
Dataset1	-99967	-99945	Dataset4	-1270785	-1270379
Dataset2	-355088	-354944	Dataset5	-902077	-900875
Dataset3	-383524	-383108	Dataset6	-541019	-540334

4.3 Parallel Performance Limits

The general parallel performance limits of RAxML have already been outlined in Section 3.2. At this point we discuss the parallel performance limits of the base method by example of Dataset3 and Dataset6 which initially appear to yield suboptimal speedups and show that those values are near-optimal. We measured the number of subproblems as well as the inference time per subproblem for one Rec-I-DCM3 iteration on those Datasets. The main problem consists in a significant load imbalance cauused by subproblem sizes. The computations for Dataset3 comprise 19 subproblems which are dominated by 3 inferences that require more than 5.000 seconds (maximum 5.569 seconds). We determined the optimal schedule of those 19 subproblems on 15 processors (since 1 processor

serves as worker) and found that the maximum inference time of 5.569 seconds is the limiting factor, i.e. the minimum execution time for those 19 jobs on 15 processors is 5.569 seconds. With this data at hand we can easily calculate the maximum attainable speedup by dividing the sum of all subproblem inference times through the minimum execution time ($37353secs/5569secs = 6.71$) which corresponds to our experimental results. There is no one-to-one correspondence since the values in Table 2 are average values over several iterations and three runs per dataset with different guide trees and decompositions.

The analysis of Dataset6 shows a similar image: there is a total of 43 subproblems which are dominated by 1 long subtree computation of 12.164 seconds and three smaller ones ranging from 5.232 to 6.235 seconds. An optimal schedule for those 43 subproblems on 15 processors shows that the large subproblem which requires 12.164 is the lower bound on the parallel solution of subproblems. The optimal speedup for the specific decomposition on this dataset is therefore $63620secs/12164secs = 5.23$.

5 Conclusion and Future Work

In this paper we have introduced P-Rec-I-DCM3(RAxML) for inference of large phylogenetic trees with ML. Initially, we have shown that Rec-I-DCM3(RAxML) finds better trees than stand-alone RAxML. The parallel implementation of P-Rec-I-DCM3 significantly reduces response times for large trees and significantly improves final tree quality. However, the scalability and efficiency of P-Rec-I-DCM3 still need to be improved. We have discussed the technical as well as algorithmic problems and limitations concerning the parallelization of the *global method* and the load imbalance within the *base method*. Thus, the development of a more scalable parallel algorithm for *global* optimization with RAxML and a more thorough investigation of subproblem load imbalance constitute main issue of future work. Nonetheless, Rec-I-DCM3(RAxML) currently represents one of the fastest and most accurate approaches for ML-based inference of large phylogenetic trees.

References

1. Additional on-line material:. www.ics.forth.gr/~stamatak.
2. M.J. Brauer, Holder M.T., Dries L.A., Zwickl D.J., Lewis P.O., and Hillis D.M. Genetic algorithms and parallel processing in maximum-likelihood phylogeny inference. *Molecular Biology and Evolution*, 19:1717–1726, 2002.
3. B. Chor and T. Tuller. Maximum likelihood of evolutionary trees is hard. In *Proc. of RECOMB05*, 2005.
4. C. Coarfa, Y. Dotsenko, J. Mellor-Crummey, L. Nakhleh, and U. Roshan. Prec-i-dcm3: A parallel framework for fast and accurate large scale phylogeny reconstruction. Proc. of HiPCoMP 2005, to be published.
5. Robert C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Research*, 32(5):1792–1797, 2004.

6. J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.
7. S. Guindon and Gascuel O. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst. Biol.*, 52(5):696–704, 2003.
8. D. Huson, S. Nettles, and T. Warnow. Disk-covering, a fast-converging method for phylogenetic tree reconstruction. *J. Comp. Biol.*, 6:369–386, 1999.
9. Camin J. and Sokal R. A method for deducing branching sequences in phylogeny. *Evolution*, 19:311–326, 1965.
10. M. Kallerjo, J. S. Farris, M. W. Chase, B. Bremer, and M. F. Fay. Simultaneous parsimony jackknife analysis of 2538 *rbcL* DNA sequences reveals support for major clades of green plants, land plants, seed plants, and flowering plants. *Plant. Syst. Evol.*, 213:259–287, 1998.
11. T.M. Keane, Naughton T.J., Travers S.A.A., McInerney J.O., and McCormack G.P. Dprml: Distributed phylogeny reconstruction by maximum likelihood. *Bioinformatics*, 21(7):969–974, 2005.
12. D. La, B. Sutch, and D. R. Livesay. Predicting protein functional sites with phylogenetic motifs. *Prot.: Struct., Funct., and Bioinf.*, 58(2):309–320, 2005.
13. S.V. Le, Schmidt H.A., and Haeseler A.v. Phynav: A novel approach to reconstruct large phylogenies. In *Proc. of GfKl conference*, 2004.
14. B. Maidak. The RDP (ribosomal database project) continues. *Nucleic Acids Research*, 28:173–174, 2000.
15. B.M.E. Moret, U. Roshan, and T. Warnow. Sequence length requirements for phylogenetic methods. In *Proc. of WABI'02*, pages 343–356, 2002.
16. K. Rice, M. Donoghue, and R. Olmstead. Analyzing large datasets: *rbcL* 500 revisited. *Systematic Biology*, 46(3):554–563, 1997.
17. U. Roshan, B. M. E. Moret, T. Warnow, and T. L. Williams. Rec-i-dcm3: a fast algorithmic technique for reconstructing large phylogenetic trees. In *Proc. of CSB04*, Stanford, California, USA, 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. I.N. Shindyalov, Kolchanov N.A., and Sander C. Can three-dimensional contacts in protein structures be predicted by analysis of correlated mutations? *Prot. Engng.*, 7:349–358, 1994.
20. A. Stamatakis, Ludwig, T., Meier, and H. Raxml-iii: A fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21(4):456–463, 2005.
21. A. Stamatakis, Ludwig T., and Meier H. Parallel inference of a 10.000-taxon phylogeny with maximum likelihood. In *Proc. of Euro-Par2004*, pages 997–1004, 2004.
22. C. Stewart, Hart D., Berry D., Olsen G., Wernert E., and Fischer W. Parallel implementation and performance of fastdnaml - a program for maximum likelihood phylogenetic inference. In *Proc. of SC2001*, 2001.
23. T.L. Williams. The relationship between maximum parsimony scores and phylogenetic tree topologies. In *Tech. Report, TR-CS-2004-04,*. Department of Computer Science, The University of New Mexico, 2004.
24. J. Wuyts, Y. Van de Peer, T. Winkelmans, and R. De Wachter. The European database on small subunit ribosomal RNA. *Nucl. Acids Res.*, 30:183–185, 2002.