Parallelization of Phylogenetic Tree Inference using Grid Technologies

Yo Yamamoto[1], Hidemoto Nakada[1,2], Hidetoshi Shimodaira[1], Satoshi Matsuoka[1,3]

[1] Department of Mathematical and Computing Sciences, Tokyo Institute of Technology
2-12-1 Oo-okayama, Meguro-ku, Tokyo, Japan 152-8550
yamamoto@matsulab.is.titech.ac.jp, hide-nakada@aist.go.jp, shimo@is.titech.ac.jp, matsu@is.titech.ac.jp
http://www.is.titech.ac.jp
[2] National Institute of Advanced Industrial Science and Technology
Grid Technology Research Center, Central2, 1-1-1 Umezono, Tsukuba, Ibaraki 305-8568 Japan
[3] National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-ku, Tokyo, Japan 101-8430

Abstract. The maximum likelihood method is considered as one of the most reliable methods for phylogenetic tree inference. However, as the number of species increases, the approach quickly loses its applicability due to explosive exponential number of trees that need to be considered. An earlier work by one of the authors [3] demonstrated that, by decomposing the trees into fragments called splits, and calculating the individual likelihood of each (small) split and combining them would result in a very close approximation of the true maximum likelihood value, as well as achieving significant reduction in computational cost. However, the cost was still significant for a practical number of species that need to be considered. To solve this problem, we further extend the algorithm so that it could be effectively parallelized in a Grid environment using Grid middleware such as Ninf and Jojo, and also applied combinatorial optimization techniques. Combined, we achieved over 64 times speedup over our previous results in a testbed of 16 nodes, with favorable speedup characteristics.

1 Introduction

All form of life today on earth originated from a common biological ancestor; so, any species may be placed as some leaf node of some gigantic phylogenetic tree. One valuable endeavor is to infer a phylogenetic tree given a set of various species to determine how the individual species have exactly evolved and relate to each other during the course of evolution, in particular when a particular branching has occurred given a pair of different species. Such research is quite important to reveal the mechanism of how evolutions have and will occur for various life forms.

Traditional biology mostly inferred the phylogenetic relationships amongst the species by their external features. However, such comparisons often tend to lack precision and objectiveness, and in fact sometimes lead to inconsistent results. With the discovery of DNA, it is now becoming possible to infer phylogenetic trees using mathematical models of evolution constructed on DNA sequences. However, in practice straightforward inference algorithms built on such models have substantial computational complexity, and have remained applicable only to very small problems.

Based on our past work that aimed to reduce the complexity in tree inference without losing precision [3], we further improve the algorithm by applying both numerical optimization and parallelization techniques on a cluster/Grid environment., using task parallel Grid middleware Ninf[2] and Jojo[1]. We obtained nearly 64-fold speedup over our earlier results as a combined effect of both on a small cluster test environment of 16 nodes, allowing us to scale the problem significantly.

2. Inferring Phylogenetic Trees – The Complexity Problem

A sample phylogenetic tree is illustrated in Figure 1. The maximum likelihood method that will compute such a tree will compute the likelihood value of x_k *at* a locus *k*, and consider the product of all such likelihood $\prod_k L(x_k)$ as the likelihood value induced from the particular DNA sequences. $L(x_k)$ will be obtained typically via a non-linear optimization process involving considerable iterations, and as will be computationally non-trivial, as shown in Table 2. After obtaining all the likelihood values of candidate phylogenetic trees, we consider the one with the largest likelihood value to be most trustworthy. However, the number of phylogenetic trees is quite large in itself, or more precisely, for *n* specifies the number of trees is $((2n-5)!)/(2^{n-3}(n-3)!) = O(2^n n!)$. As a result, computing the likelihood values for all possible candidate trees becomes quickly impractical, even for a relatively small *n*.

To cope with such massive computational complexity, one of the authors proposed an approximate method for computing the likelihood value of a given tree. We call a branch of a phylogenetic tree a *split*, and given n species we can divide a given phylogenetic tree into $(n-3)$ set of splits. We then compute the likelihood values of given splits using the maximum likelihood method, and derive an approximate value of the likelihood value using matrix manipulations. This method has considerable computational complexity advantage without losing much precision. Since the total number of possible splits is $2^{n-1}-(n+1) = O(2^n)$, we can significantly reduce the overall computational cost. However, the number of phylogenetic trees is still significant, and this method, although a definite improvement, still was too expensive of realistic values of *n*.

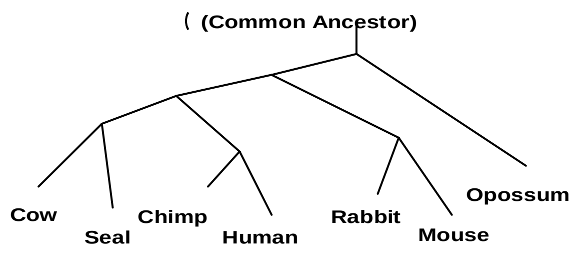

Fig. 1. An Example Phylogenetic Tree

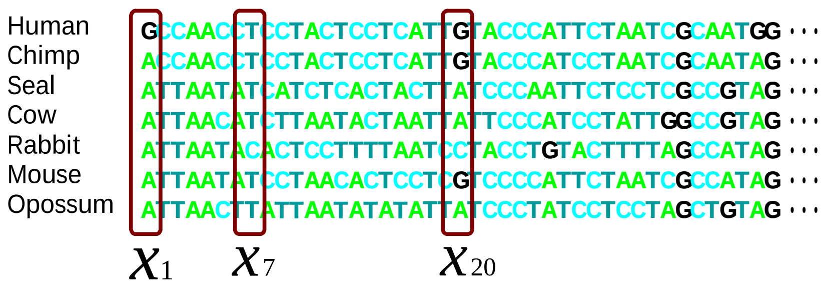
Human GCCAACCTCCTACTCCTCATTGTACCCATTCTAATCGCAATGG ···
Chimp ACCAACCTCCTACTCCTCATTGTACCCATCCTAATCGCAATAG ···
Seal ATTAATATCATCTCACTACTTATCCCAATTCTCCTCGCCGTAG ···
Cow ATTAACATCTTAATACTAATTATTCCCATCCTATTGGCCGTAG ···
Rabbit ATTAATACACTCCTTTTAATCCTACCTGTACTTTTAGCCATAG ···
Mouse ATTAATATCCTAACACTCCTCGTCCCCATTCTAATCGCCATAG ···
Opossum ATTAACTTATTAATATATATTATCCCTATCCTCCTAGCTGTAG ···
X_1
X_7
X_{20}

Fig. 2. DNA Sequence, and computing of the likelihood values.

3. Overview of our Proposed Improvements to the Split Method

Our improvements over the previous proposal are twofold. One is to apply combinatorial optimization techniques to reduce the search space for the trees. Another is to parallelize the search effectively over the Grid using appropriate task-parallel Grid middleware, Ninf[2] and Jojo[1]. The resulting program was shown to execute efficiently and with significant speedup, even for a relatively small number of nodes on a small-scale cluster. Figure 3 shows the overall workflow; here, we see that the program largely consists of two phases, the first phase being “computing the likelihood value of each split using the maximum likelihood method”, and the second phase being the “combining the splits and searching the optimal results using combinatorial optimization techniques and their parallelization”. The former will perform parameter-sweep parallelization of likelihood values of each possible split, either sequentially or in parallel on the Grid, and output the results in files for the second phase. The second phase in turn will either directly obtain the likelihood values of all combinations of splits, or use combinatorial optimization techniques such as branch-and bound or simulated annealing, and obtain the optimal likelihood value from the “more likely” candidates, again possibly in parallel on the Grid.

For both phases, we parallelize the computation using master-worker scheme, and implement the former using the Ninf GridRPC system, whereas for the latter we perform further hierarchical master-worker parallelization using a Java Grid parallelization system Jojo. There are various reasons we employ two different Grid middleware systems; the primary reason for using Ninf GridRPC is that, it is easy to integrate existing maximum likelihood numerical packages, while the reason we employ Jojo for the second phase is that, the latter involves hierarchical parallelization, in particular branch-and-bound computation. From a pure Grid middleware research perspective it is also interesting to investigate how the two different middleware will interoperate smoothly on the Grid.

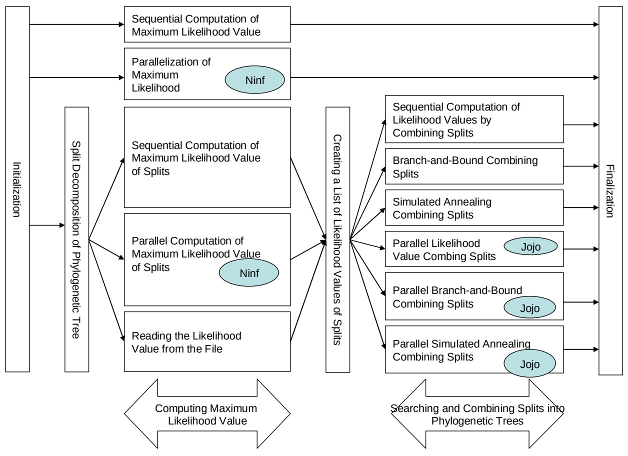
Initialization
Sequential Computation of Maximum Likelihood Value
Parallelization of Maximum Likelihood
Ninf
Split Decomposition of Phylogenetic Tree
Sequential Computation of Maximum Likelihood Value of Splits
Parallel Computation of Maximum Likelihood Value of Splits
Ninf
Reading the Likelihood Value from the File
Creating a List of Likelihood Values of Splits
Sequential Computation of Likelihood Values by Combining Splits
Branch-and-Bound Combining Splits
Simulated Annealing Combining Splits
Parallel Likelihood Value Combing Splits
Jojo
Parallel Branch-and-Bound Combining Splits
Jojo
Parallel Simulated Annealing Combining Splits
Jojo
Finalization
Computing Maximum Likelihood Value
Searching and Combining Splits into Phylogenetic Trees

Fig. 3. The Overall Workflow of Deriving the Phylogenetic Tree with Maximum Likelihood Value.

4. Optimizing Phylogenetic Tree Inference

In order to reduce the number of candidate phylogenetic trees, we employ combinatorial optimization techniques.

4.1. Using Branch-and-Bound

Since each phylogenetic tree corresponds to $(n-3)$ sets of splits, we obtain a new tree by combining a split onto a star-shaped phylogenetic tree one by one, as shown in Figure 4. Since there are multiple ways how split can be combined at each stage, the search space branches out into a search tree as in Figure 4, with the leaves being the candidate phylogenetic tree. We then apply branch-and-bound technique onto this search tree, allowing us to prune the search space significantly with an appropriate bounds function.

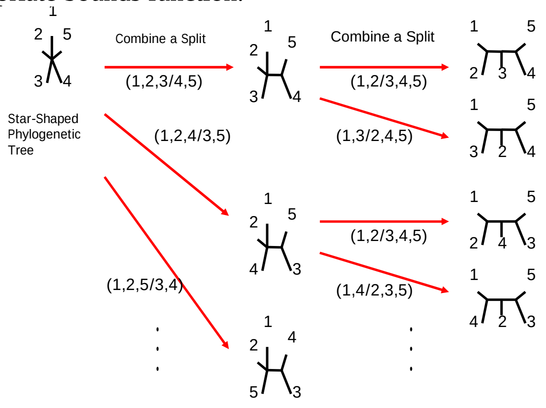

Fig. 4. Formulating a Search Tree of Combining Splits

In order to prune the branches, we compute the upper bound of the likelihood value for each node in the tree, and compare the value against the current solution; if the upper bound is greater, we continue the search by expanding the node; otherwise, we prune the branch of the search tree under that node.

As an upper bound, we employ the combined likelihood value where we combine all possible splits onto that (pair of splits) node.

4.2. Using Simulated Annealing

For simulated annealing, we obtain the neighboring solutions based on splits, as we see in Figure 5. Firstly, from the set of splits that signifies the current solution, we arbitrarily remove one of the splits. Then, of the three possible splits that could be combined with the current set, we remove the split that would result in idempotent return to the original before split removal. Then, from the remaining two we pick one at random, and combine with the set of splits, deriving the neighboring solutions. The "cooling" function we employed was, given the cooling parameter α we simply perform exponential degradation $T_{next} = \alpha T_{current}$ $(0 << \alpha < 1)$.

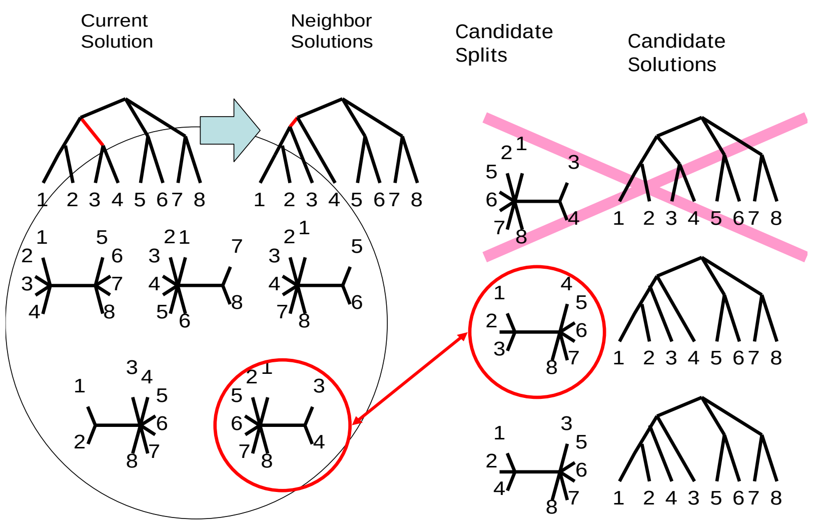

Fig. 5. Creating Neighboring Solutions and Candidates in the Simulated Annealing Scheme

5. Parallelizing Phylogenetic Tree Inference on the Grid

5.1. Using Simulated Annealing

The computation in the first phase continues in a simple master worker style. The master first creates a pool of phylogenetic trees or splits subject to further computation, and sends them as jobs off to the worker nodes on the Grid one by one. The worker computes the maximum likelihood and returns the value; the worker aggregates the returned result and resends another job to an idle worker, until the pool of trees and/or splits is exhausted.

5.2. Parallelizing the Split Method Directly

The second phase can also be parallelized in a straightforward fashion in a similar manner using master-worker style computation. This time the master sends the phylogenetic tree and the corresponding split pair to the workers. The workers in turn computes the approximate likelihood values of the combined phylogenetic tree based on the likelihood values obtained in phase one, and returns the result to the master, The master keeps track of the process, continuing until all the phylogenetic trees that can be generated are covered, and picks the tree with the largest likelihood value as the result.

5.3. Efficient Parallelization of the Split Method using Branch-and-Bound

With branch-and-bound, parallelization is performed somewhat differently for phase two. However, a common problem with parallelizing branch-and-bound is that, there could be considerable load imbalance depending on how the search tree is divided, and moreover, some computation may go to waste if the bound value of some branch turns out to prune a computation ongoing on some other processor. The shape and the depth of subtrees of a search tree may greatly differ amongst one another, and whether or not a search should be conducted on a subtree is runtime dependent. As such, naïve master-worker subdivision of (sub) search tree may turn out to be quite inefficient.

In order to avoid this problem, we set a limit on the number of "problems" (i.e., computing the likelihood value of each split) individual workers will compute for each subtree. The master maintains a pool of problems, and distributes each one by one to the workers. The worker then proceeds to solve the problems in the manner similar to Section 5.2. If the number of problems that the worker has solved exceeds some threshold value, the problem is returned to the master. The master in turn re-adds the problem to the pool for subsequent re-allocation to some worker.

5.4. Parallelizing Simulated Annealing

For parallelizing simulated annealing, we employed the replica exchange method as outlined below. Each worker maintains an independent simulated annealing process, each with a different temperature parameter. At some periodic *exchange interval* the worker sends the current likelihood value to the master, and requests for exchanging the temperature value. Let us label this worker *i*. When the master receives this request, the master notifies worker *j*, where $j < i$ and holds the maximum temperature value, that an exchange request has been made. The worker *j* in turn judges whether the exchange of the temperature should take place according to the temperature and the likelihood value it has received. If it decides to accept the exchange it performs the temperature exchange and notifies the master its previous temperature and the current likelihood value; the master then notifies worker *i* that the exchange was successful, the worker *i* performs its own exchange internally, and the entire exchange process completes. Otherwise, if the worker *j* decides to deny the request, it notifies the master who in turn notifies worker *i* that the exchange was unsuccessful. Irrespective of whether the exchange was successful or not, the workers continue with their search until the next exchange period is encountered.

6. Evaluation of Our Proposed Scheme

6.1. Evaluation Criteria and the Target Problem

As the evaluation criteria, we employed and measured the followings:

- **Evaluating the original approximation algorithm using splits**: We initially investigate the effect of our original scheme of deriving the approximate likelihood value by combining splits to form the phylogenetic tree, by comparing its precision and compute cost
- **Evaluating the reduction of the search space using combinatorial optimization techniques:** We next evaluate how much the search space has been reduced by employing branch-and-bound and simulated annealing techniques
- **Evaluating the parallelization efficiency and scalability:** Although in theory master-worker computation could approach near-perfect speedup, in reality we may observe loss in efficiency as we scale the problem larger due to various factors including communication overhead between the master and the worker, as well as the load of the master increasing and becoming the sequential bottleneck. The metrics we employ are firstly the speedup value due to parallelization, i.e.,

$$Speedup = \frac{T_{serial}}{T_{parallel}}$$

  and in particular we measure the parallelization *overhead* as when we have just one master and one worker. We also compute the scalability as:

$$Scalability = \frac{Speedup}{N_{workers}}$$

  where $N_{workers}$ is the total number of workers.

We employed paml[4] as the program that computes the likelihood value given a particular base sequence. For our sample problem we used the following 9 species: seal, cow, rabbit, opossum, mouse homo-sapience, dugong, armadillo, and rat. The sequence data for each species are those for mitochondria downloaded from NCHI, and the sequence length is 3392.

6.2. Evaluation Environment

As a preliminary evaluation environment, we employed a cluster as a controlled platform rather than to employ a full-fledged distributed Grid. The employed Abacus cluster consists of dual Athlon MP 2000+ (1.67 Ghz) nodes with 1GigaByte of memory, and interconnected via a 100Base-T network. Both the master and the worker have been allocated on the nodes, and we measured using 2, 4, 8, and 16 nodes.

6.3. Evaluating the original approximation algorithm using splits

Figure 6 shows the graph of the results of evaluating our original approximation algorithm using splits. We employed 6 species from our 9, and from the total of possible 105 phylogenetic trees, on the horizontal axis we plot the exact maximum likelihood value, versus the approximated value computed by combining splits. As we observe from the graph, most of the values lie on the line $y = x$: as such, we observe that our original approximation method is quite good.

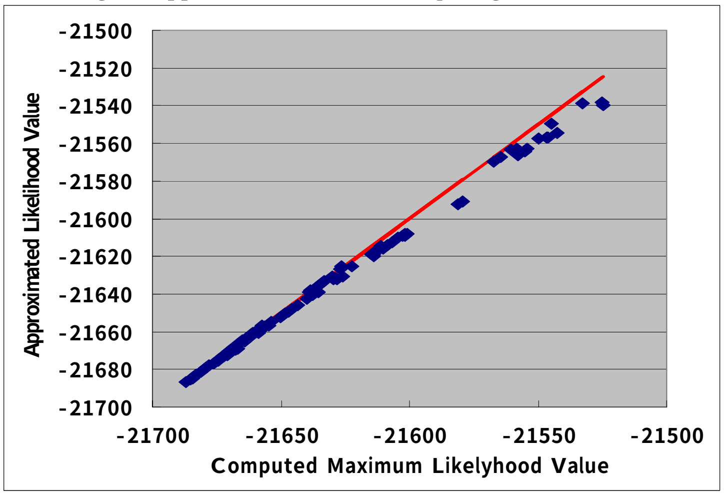

Fig. 6. Comparing Approximated Likelihood Value with the Real Maximum Likelihood Value

Table 1 shows the effect on the computational costs using our approximation method. For all possible phylogenetic trees for *n* species, we compare the actual computation time of the approximation method, denoted as T_{comp}, versus the projected computation time using the maximum likelihood method, denoted as T_{paml} (computed as $T_{ave} \times T_{ave}$, where T_{ave} is the sampled average value of computing the maximum likelihood value). As we observe in the table, we drastically reduced the overall compute time.

Table 1. Effect of Computational Cost Reduction Using the Approximation Medhod.

n	Ntree	Tave (sec)	Tpaml	Tcomp
5	15	37	9 min 30 sec	3 min 7 sec
6	105	85	2 hours 30 min	6 min 55 sec
7	945	149	1 day 15 hours	35 min 22 sec
8	10,395	241	29 days	2 hours 44 min
9	135,135	330	1 year 5 months	18 hours 41 min

The approximation method internally consists of two phases, where we compute the maximum likelihood value of each split, and the second phase where we combine the splits to form each phylogenetic tree. Figure 7 shows the breakdown of the compute time for these two phases. As we can see, the time to combine the split becomes more dominant as *n* grows larger. This is because the number of splits is $O(2^n)$ whereas the number of possible phylogenetic trees is $O(2^n n!)$. Based on this observation we simply parallelized the first phase whereas we performed parallelization as well as combinatorial optimization for the second phase.

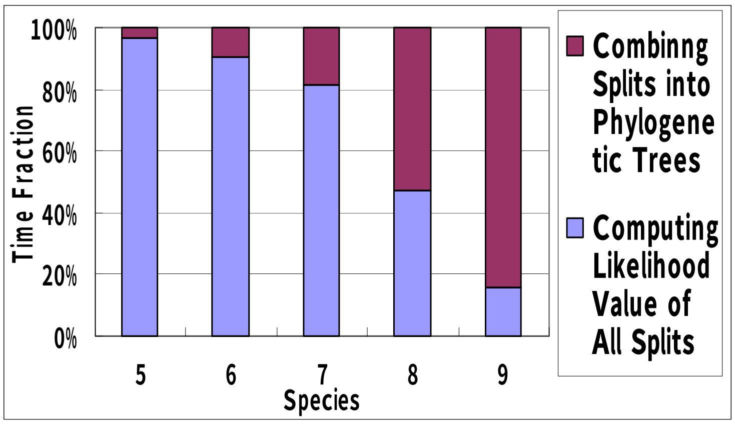

Fig. 7. Time Fraction of Two Phases in the Approximation Algorithm

6.4. Evaluating the reduction of the search space using combinatorial optimization techniques

6.4.1. Branch-and-Bound

For branch-and-bound, we measured how much pruning of the search tree is achieved. Since on every interior search nodes we need to combine all possible splits combinable with the current split that represent the search status, the interior nodes take just as much time to compute the likelihood value as the leaves; because of this we need to account for this cost and not just account for the computational cost at the leaves. In total, we were able to prune the search space by 83.4%, 93.5%, and 95.3% for 7, 8 , and 9 species, respectively.

6.4.2. Simulated Annealing

For simulated annealing, the number of times the search is repeated depends on the termination condition. Here, for benchmarking purposes, we precompute the likelihood values of all the phylogenetic trees beforehand, and have the termination condition be such that the result falls within 1% of the precomputed results for 10 consecutive times. Also, since there will be effects of randomness in the results, we perform the experiment 3 times and take the average of the results (Note that these are purely for evaluation purposes of the obtained results, and not something to be used in practice.). We alter the initial temperature and the cooling parameters in various experiments.

For 7 species, we found that, when we set the initial temperatures to be low, the number of search repetitions will be smaller, but we also have had to enlarge the cooling parameters, or otherwise we may not achieve convergence in the results. Experimenting on various parameters, we found that we could converge and reduce the search space by up to 95.2%, similar to the results obtained with branch-and-bound.

6.5. Evaluation of Scalability with Parallelization

On evaluating simple parallelization of maximum likelihood value computation for 5-6 species, we found the overhead to be about 12% for 6 species. This overhead is by no means small, and increases with larger *n* instead of decreasing, as one may expect normally since the granularity does not decrease with a larger *n* We currently consider that the overhead is largely due to I/O of the phylogenetic tree itself, and working to resolve the issue. The results that follow must be regarded with this issue in mind.

For scalability, we obtained approximately 6.5 to 7.5 speedup with 8 workers, and 12.7 speedup with 16 workers. Given that the overhead is 12%, these are nearly ideal results.

For parallelizing our approximation scheme without combinatorial optimizations, the overhead ranged from 30% for 5 species, to about 5% for 8 species. Figure 8 shows the scalability graph, where we obtained 10.5 times speedup for 8 species on 16 nodes.

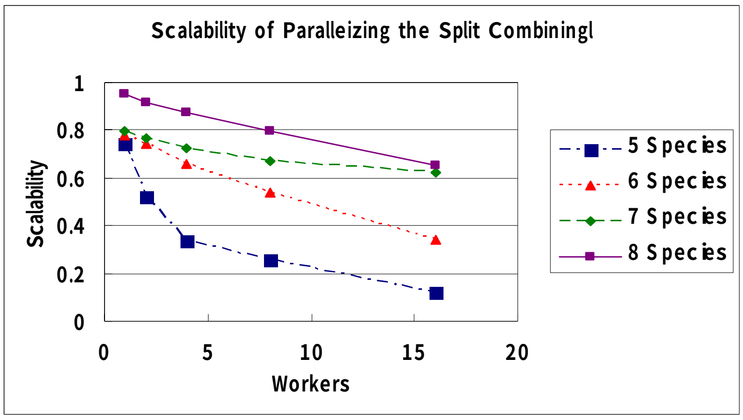

Fig. 8. Scalability of Parallelizing the Combining of the Splits in our Original Approximation Algorithm.

6.5.1. Evaluating Parallelization under Branch-and-Bound

We varied the number of species by 5-9 for evaluating parallelized branch-and-bound. The overhead due to parallelization is less that 3-4% for over 5 species, indicating effective parallelization. Scalability is quite excellent as seen in Figure 9. We need to conduct experiments with a significantly larger *n* to observe scalability of our parallelization on a larger number of nodes on the Grid, however.

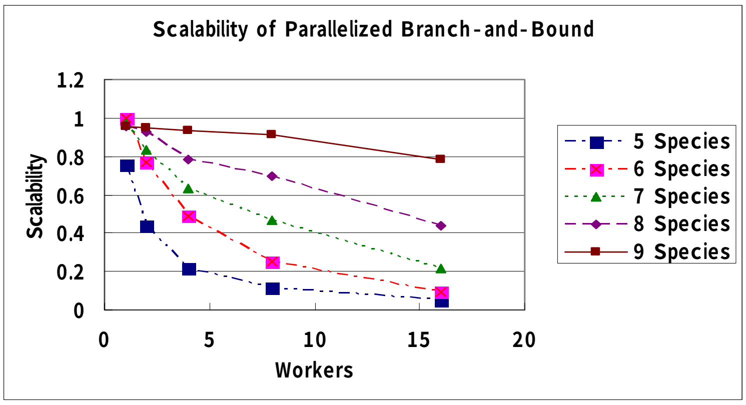

Fig. 8. Scalability of Parallelizing the Branch-and-Bound Optimization

6.5.2. Evaluating Parallelization of Simulated Annealing (Replica Exchange Method)

Finally, for simulated annealing parallelized with replica exchange method, we set the maximum temperature values to 5, 10, 20, 50, 100, 200, and 500, and the minimum temperature to be 1. Each worker was assigned an initial temperature value so that the intervals of the temperature values will have constant ratio with its neighbors, and cover the range from the minimum to the maximum. For example, if the maximum temperature is 8, and the number of workers is 4, the ratio will be 2, and the assigned values will be 1, 2, 4, and 8. We experimented with 4 to 16 workers and 7 species, with the the termination condition be such that if one of the workers satisfies the same condition as the sequential case, the entire system is terminated.

Figure 10 shows the results: here, we observe that (as expected) the replica exchange method derives no speedup, but rather contributes to stability of the results convergence. Irrespective of the initial temperature value, the results converge after about 60 iterations, achieving 94% pruning of the search space. Although subject to benchmarking in a larger system, the initial results are favorable in that the user of our scheme may be relieved of truing the parameters appropriately to achieve fast convergence, when simulated annealing is advantageous over branch-and-bound.

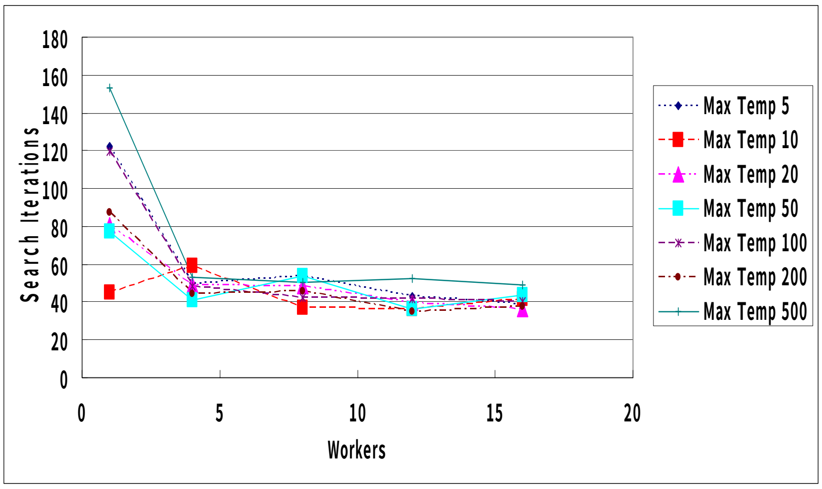

Fig. 8. Convergence Characteristics of Simulated Annealing / Replica Exchange Method

7. Conclusion and Future Work

We demonstrated that, by combining our approximation scheme of the likelihood value of phylogenetic trees with appropriate combinatorial optimization techniques and parallelization techniques on the Grid, we obtain substantial speedups with good efficiency, pruning the search space as much as 95% for both branch-and-bound as well as simulated annealing techniques. The net effects of all the approximations and optimizations was 64 times over our original approximation method, or over 50,000 fold speedup over the straightforward sequential algorithm for obtaining maximum likelihood value for each candidate tree, with 16 processors. This is a promising result that allows comparison of fairly large number of *n* in a realistic timeframe.

As a future work, we need to further investigate methods for coping with scaling up the computation for a larger *n*. In particular, we need to experiment with other combinatorial optimization techniques, such as Genetic Algorithm, and/or other algorithms for computing the likelihood values more efficiently. We also need to reduce the parallelization overhead as well as matching the more hierarchical nature of the resources on the Grid. Here, the hierarchical organization features of Jojo could be of good help, but we need to validate the scalability in a real Grid. Another issue under a practical setting is fault tolerancy, which has not been built into our system. Since master-worker style computation is somewhat amenable to easier checkpointing, we are planning to use some of the checkpoint as well as recovery features in the new versions of Ninf and Jojo. Finally, we need to make both our code as well as our environment available, possibly in the form of portals so that computed results could be systematically stored for access by phylogenetic researchers.

Acknowledgements

This research was partially funded by the Japanese ACT-JST "Terascale large-scale optimization using commodity Grid technologies" project, as well as the Japanese Scientific Grant-in-Aid (A) 14702061.

References

[1] Hidemoto Nakada, Satoshi Matsuoka, and Satoshi Sekiguchi. "A Java-Based Programming Environment for Hierarchical Grid: Jojo", in *Proceedings of IEEE Computing in Clusters and the Grid (CCGrid) 2004*, the IEEE Press, April, 2004.

[2] Hidemoto Nakada, Mitsuhisa Sato, and Satoshi Sekiguchi. *"Design and Implementation of Ninf: Towards a Global Computing Infrastructure"*. In *Future Generation Computing Systems, Metacomputing Issue*, Vol. 15, pp.649-658, 1999.

[3] Hidetoshi Shimodaira. "Multiple Comparisons of Log-Likelihoods and Combining Nonnested Models with Applications to Phylogenetic Tree Selection", in *Comm. In Statistics, Part A-Theory and Meth.*, Vol. 30, pp. 1751-1772, 2001.

[4] Z. Yang. "Paml: A Program Package for Phylogenetic Analysis by Maximum Likelihood", in *CABIOS*, Vo. 13, pp. 555-556, 1997.