Parallel Multiple Sequence Alignment with Local Phylogeny Search by Simulated Annealing*

Jaroslaw Zola[1], Denis Trystram[1], Andrei Tchernykh[2], Carlos Brizuela[2]

[1]Laboratoire ID–IMAG
51 av. J. Kuntzmann
38330 Montbonnot, France
{zola,trystram}@imag.fr

[2]Computer Science Department
CICESE Res. Center
Ensenada, BC, 22860, Mexico
{chernykh,cbrizuel}@cicese.mx

Abstract

The problem of multiple sequence alignment is one of the most important problems in computational biology. In this paper we present a new method that simultaneously performs multiple sequence alignment and phylogenetic tree inference for large input data sets. We describe a parallel implementation of our method that utilises simulated annealing metaheuristic to find locally optimal phylogenetic trees in reasonable time. To validate the method, we perform a set of experiments with synthetic as well as real–life data.

1 Introduction

Multiple sequence alignment (MSA) is undoubtedly the most commonly performed task in the computational biology. It is an essential prerequisite of many other complex problems like, for example, database search or phylogenetic analysis [1, 12]. Unfortunately, finding an accurate multiple alignment is a hard optimisation problem. Firstly, because it is difficult to provide a formalisation which would be satisfactory from the biological viewpoint. Secondly, having a good model usually means it is algorithmically very hard to find the best (or optimal) alignment. Indeed, the *Generalized Tree Alignment Problem* (GTA), which seems to be the most accurate formalisation of MSA, has been shown to be *Max–SNP–Hard* [13].

Multiple sequence alignment and phylogenetic tree reconstruction are typically considered separately, however, both these problems are directly linked. For instance, most of the character based approaches to the phylogeny reconstruction (e.g. minimum parsimony or maximum likelihood methods) require MSA as an input data. Moreover, the quality of the trees reconstructed by these methods depends on the quality of the input alignment [11]. On the other hand, looking for similarities among sequences we should be aware of the fact, that they are a result of some complex evolutionary process. Thus, in the optimal case, we should know their relationship which is described by some phylogenetic tree.

In our previous contributions [19, 26] we have reported a new heuristic to solve GTA problem in parallel. In this paper, we extend this approach by adding a simulated annealing (SA) metaheuristic to find optimal partial phylogenetic trees during a phylogenetic analysis step of our method. The results we report show that application of SA allows running time of our software to be decreased by several orders of magnitude while preserving a good quality of final solutions.

The remainder of this paper is organised as follows: In Section 2 we provide a brief overview of the *Parallel PhylTree* method. Section 3 contains description of our strategy to infer local phylogenetic tree by SA. In Section 4 we show some experimental results with actual biological data. Finally, possible extensions of this work are discussed in Section 5.

2 Parallel multiple sequence alignment

2.1 The PhylTree method

The *PhylTree* method is a generic, parallel multiple sequence alignment procedure which has been proposed recently [19, 26]. As it has been presented in detail in our previous papers, we provide here only its basic concepts with no technical details.

The *PhylTree* method was designed to build a multiple alignment and the corresponding phylogenetic tree simultaneously. It can be characterised as an iterative clustering method whose principle is to group closely related sequences [10]. It consists of two successive phases: first it generates a distance matrix for all input sequences (based on the all–to–all pairwise alignments). Then, it searches for the optimal partial solutions which are later combined to obtain the final phylogenetic tree and multiple alignment.

The method uses two principles: *neighbourhood* and *cluster*. A *neighbourhood* is a set of k closely related taxa, where k is an input parameter of the *PhylTree* procedure and is a constant integer value (typically, $k \geq 4$, not larger than 10). A *cluster* is a group of $m \leq k$ taxa which creates a part of the final phylogeny. To determine the clusters the *PhylTree* algorithm generates a neighbourhood for every input sequence, then it finds the best phylogenetic tree for each neighbourhood, and finally, it analyses all the found trees to extract their common subtrees. These subtrees describe a set of highly related taxa and correspond to clusters. The above process is iterative.

The accuracy of the method depends mostly on the parameter k. Increasing k will widen the search space analysed to find the clusters. Unfortunately, while this improves the accuracy of the final solution, it also increases the number of computations required to process a single neighbourhood. Basically, to process a neighbourhood we have to compute all possible $\frac{(2k-2)!}{2^{k-1}(k-1)!}$ trees to find the optimal one.

An important property of *PhylTree* is its generality, which means that the method can use different alignment algorithms and different scoring functions. Moreover, it is possible to use various definitions of a neighbourhood and neighbourhoods of variable size (a variant called *QuickTree* has been proposed as a way of rapid solving large instances with decreased accuracy).

2.2 Parallelisation scheme

Due to high time complexity of the *PhylTree* method we have implemented its parallel version based on the decentralised cache of alignments [26].

The *Parallel PhylTree* method takes advantage of the inherent parallelism of the original algorithm, and it utilises master–worker architecture combined with the cache system working in the peer–to–peer manner.

The *Parallel PhylTree* method proceeds as follows (see Figure 1): First, the master processor reads the input sequences and broadcasts them to all workers. Next, each worker receives a part of the distance matrix to compute. At this stage, we use the *guided self–*
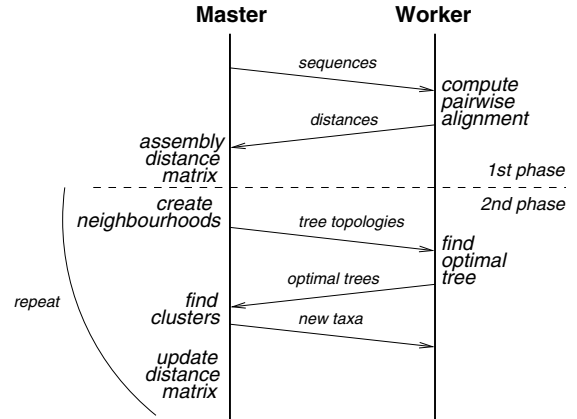

Figure 1. General scheme of the *Parallel Phyl-Tree* **method.**

scheduling strategy [20]. Such approach allows the load imbalance imposed by the heterogeneous environment to be minimised. Please note that heterogeneity may be the result of high diversity in the length of input sequences, and not only differences in the hardware architectures. Processing the distance matrix, a worker analyses its efficiency by measuring the number of base pairs aligned per second. Thanks to this analysis, the master node can rank all the workers accordingly to their efficiency. In the second phase, only the processing of the neighbourhoods is parallelised. That is: at each iteration, the master determines the neighbourhood sets, and then it starts to generate all possible tree topologies for each neighbourhood. Each worker receives its part of the generated topologies and looks for one with the highest alignment score. At this point we assume that for each neighbourhood an optimal partial solution must be generated. To achieve this objective, worker has to compute MSA for every single topology. Because the number of created tree topologies for all neighbourhoods is usually much greater than the number of workers, again dynamic scheduling is used. However, at this stage, the host priorities computed in the first phase are utilised. Half of the total number of topologies is distributed proportionally to the workers' priorities. Then, the other part is distributed using the guided self–scheduling. When a worker completes its part of the computations it sends back the resulting best tree topology and the corresponding alignment to the master. To complete an iteration, the master computes the clusters and updates the distance matrix accordingly.

Since neighbourhoods of sequences may overlap, and pairwise alignments computed in the first phase of

the method are reused in the second phase, the *Phyl-Tree* method exhibits high redundancy of computations. Therefore, in the parallel version we provide decentralised cache of alignments [26]. Our cache stores all intermediate results of alignment, and is organised into content addressable network [21]. Thanks to this a significant speedup can be achieved even in a single run of the *Parallel PhylTree*.

2.3 Related work

Both multiple sequence alignment and phylogeny are of great importance to biologists, but at the same time these problems are very computationally demanding. That is why parallel and distributed programming is often used to improve the efficiency of existing bioinformatics applications.

In the last few years several popular MSA packages have been parallelised, for example DiAlign [23] or Praline [15]. Yet, most of the attention was focused on the ClustalW tool. There are numerous parallel versions of ClustalW, e.g. [5, 16, 17], designed for both shared and distributed memory architectures. However, in most cases, the parallel approach is limited to the main ClustalW algorithm. For instance, in the ClustalW-MPI [16] Li utilises *fixed–size chunk* scheduling to distribute tasks in the first stage. Next, only the third stage is parallelised, but the efficiency of this part depends on the balance of the guide tree. The reported average speedup for this part was around 4.5 for 16 CPUs. However, the overall speedup was almost linear (15.8 for 16 CPUs).

The ClustalW package performs progressive alignment, and it utilises distance based strategy (usually *neighbour–joining*) to reconstruct alignment guide tree. The quality of the reconstructed tree is usually very poor, and in the terms of protein alignment ClustalW is outperformed by many other packages, e.g. MUSCLE [6]. On the other hand, ClustalW achieves very good results in the case of RNA alignments [7], thus it still remains very useful.

3 Local phylogeny inference

As we already indicated, during the second phase of computations the *PhylTree* tries to find the optimal phylogenetic trees for a set of previously determined neighbourhoods. This step is the most time consuming part of the method: To evaluate a single tree an accompanying multiple sequence alignment must be computed. Here we assume that any alignment scoring function suitable for a given type of input data can be applied. For example, in our current implementation

we provide scoring functions based on the parsimony criterion with linear gap penalties but also basic sum of pairs function, and consistency based function *COFFEE* [18].

While the ability to use various scoring schemes extends possible applications of *PhylTree*, it makes difficult to apply faster than exact trees enumeration search techniques. For instance, to find an optimal tree under parsimony criterion a branch and bound strategy could be used, whereas it is not necessarily true for other scoring functions.

3.1 The simulated annealing approach

To improve efficiency of the *PhylTree* we have decided to apply metaheuristic search approach based on simulated annealing [14]. Our solution is based on the observation that groups of closely related elements, which are recognised as *phylo–clusters*, should be detectable even when suboptimal trees for analysed neighbourhoods will be found. This is because partial trees which correspond to *phylo–clusters* contribute strongly to the score of the optimal tree for a given neighbourhood. Similar observations have been reported by other researchers, e.g. [27].

Simulated annealing (SA) is a well known and easy to use metaheuristic search strategy. The algorithm works by analogy with the physical process of annealing in which the free energy of the solid is minimised. A solution to the optimisation problem corresponds to the physical state of the material, and the cost of a solution (the value of objective function) corresponds to the free energy of the material. In our case, the solution is a partial guide tree for a given neighbourhood, and its free energy is the score of the corresponding multiple alignment.

It is worth mentioning that simulated annealing has been successfully applied in the problem of phylogeny search under the parsimony criterion [2], as well as, in the case of maximum likelihood phylogeny inference [24].

In general, there are four main components of the SA algorithm:

- *Configuration generator*, which is a method of generating a candidate solution x_{i+1} on the basis of current solution x_i.

- *Cooling scheme*, which describes the procedure of decreasing the system temperature T. This parameter has a direct influence on the probability of acceptance of backward steps (e.g. to escape local optima).

- *Metropolis–step*, in which algorithm decides if a candidate solution x_{i+1} should be accepted. If x_{i+1} has better cost than x_i it is always accepted. If not, it is accepted with a probability $P = e^{-(\Delta H/T)}$, where ΔH is the cost change.

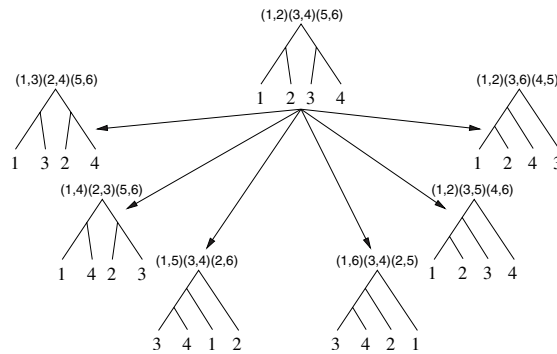- *Stopping criterion*, which determines when to finish searching.

One important property of SA, as compared to other "stronger" metaheuristics like for example genetic algorithms, is relatively smaller number of objective function evaluations. In our case, objective function evaluation means computing a multiple sequence alignment, hence, this property is of great importance.

In the *PhylTree* with SA the analysis of the neighbourhood begins with a random guide tree. The configuration generator is implemented using a bijection between rooted phylogenetic tree with k leaves and perfect matching on $2k - 2$ points [4]. Here, a perfect matching on $2l$ points is defined as a partition of these points into l two–element subsets. Having such perfect matching we can render a corresponding phylogenetic tree as follows. First, we choose a matched pair which has both elements from the set $\{1, \ldots, k\}$ which is the set of leaves. If more than one such pair is available we choose the one with the smallest element. Next, we assign to this pair a parent node which is the first available not–leaf label in the range $[k+1, 2k-2]$. From now on, this label is considered a leaf, and we continue the process. Figure 2 shows an example tree topology together with six configurations which can be generated on its basis.

Application of the matchings provides a direct way to generate a random walk in the tree search space. To obtain a new topology it is sufficient to make a transposition of two points in the matching. Obviously, such configuration generator can be implemented easily.

Our implementation of simulated annealing approach allows for use of different cooling schemes, for example, a simple model where temperature T is decreased with constant ratio $T_{i+1} = T_i - \Delta T$, or another one where $T_{i+1} = \alpha \cdot T_i, \alpha < 1$. However, experiments we have performed showed that there is no noticeable difference in the performance of various cooling schemes.

In our current implementation SA computations can be stopped by master to keep workers load balanced (see next subsection), or in other case, they proceed a fixed number of iterations, which can be defined by the user. The initial temperature of the system is set on the basis of sequence length and substitution matrix, so that proper values of the probability P in the Metropolis–step are guaranteed. Finally, the number

Figure 2. Example tree and its 6 neighbour topologies.

of iterations in the *Metropolis–step* is equal to the size of neighbourhood.

The search space decoded using perfect matchings can be described using regular, undirected graph G with vertices of degree $k \cdot (k - 3) + 2$, where k is the size of the sequence neighbourhood. Since graph G is regular it is very likely that some vertices (search space points) will be visited more than once during a single search. Subsequent evaluations of the same vertex are handled by the complex caching system [26], nevertheless they generate some overhead when retrieved from the cache storage. Moreover, unconstrained random walk in the graph G may contain cycles.

To avoid redundant evaluations of the same trees we extended SA algorithm with a naive tabu list. The list stores all points visited during SA search, and each point is encoded using only a few bytes, so that memory consumption is negligible. When a candidate solution is generated, the tabu list is checked first and if the solution is not tabu, it is evaluated and inserted into the tabu list. On the other hand, if a given solution is found on the tabu list, it is rejected by SA and another solution is generated. In the case where there are no acceptable points to move to, the temperature parameter of the SA is restored to the value from the previous step of SA, and search is restarted from some randomly chosen feasible solution. This way we avoid possible deadlocks, and we improve efficiency of the search algorithm. Please note that, while our approach differs substantially from the typical tabu search algorithm [8] (we do not restrict moves, but points, and we do not use aspiration level mechanism) it performs in a similar way. This is because each point in our search space can be reached in several ways, and restricting a point we do not restrict reachability of its neighbourhood.

3.2 Integration with *Parallel PhylTree*

To integrate the simulated annealing approach described above the parallel version of the *PhylTree* has been modified. To be more precise, the phylogenetic analysis step, which in the case of exact enumeration is based on the tree topologies distribution (see Section 2.2), has been re-implemented to allow simultaneous analysis of several neighbourhoods.

In a single iteration of the second phase, each worker receives one neighbourhood to analyse. If the number of neighbourhoods to process is less than the number of available workers, then idle nodes are assigned to perform redundant neighbourhood analysis. This means, that some of the neighbourhoods can be processed by more than one worker. The assignment of neighbourhoods to workers is performed in round–robin fashion. The obvious drawback of this approach is that load balancing is hard to maintain. To avoid the problem of load imbalance, redundant analysis of the neighbourhoods can be interrupted at any moment. As a result, phylogenetic analysis ends as soon as all unique input neighbourhoods have been processed. Resulting partial guide trees, are the best trees found.

Due to a very coarse grain of the above algorithm, in the case when the number of available processors is grater than the number of neighbourhoods to process it may be more profitable to apply a more fine grained exact search strategy (as described in Section 2.2).

Let P be the number of available workers. In the exact search approach expected time of completion of a single iteration can be expressed as follows:

$$T_e = \left\lceil \frac{n \cdot A}{P} \right\rceil \cdot t_{avg}, A = \frac{(2 \cdot k - 2)!}{2^{k-1}(k-1)!}$$

where n is the number of input sequences in a given iteration, k is the size of the neighbourhood of a single sequence, and finally t_{avg} is an average time required to analyse a single tree. In the case of SA we can assume that complexity of the search method is $O(k^3)$, thus, expected time of single iteration completion is:

$$T_{sa} = \left\lceil \frac{n}{P} \right\rceil \cdot k^3 \cdot t_{avg}$$

On the basis of the above expressions we have implemented a mechanism which dynamically changes search strategy. In every iteration the master node verifies if $T_{sa} < T_e$. If the inequality is satisfied, SA approach is utilised, otherwise workers perform exact search. Figure 3 summaries our new implementation of the *PhylTree*.

Compute distance matrix in parallel;
repeat
     Determine neighbourhoods;
     if $T_{sa} < T_e$ **then**
         Distribute neighbourhoods;
         Analyze neighbourhoods using SA;
     else
         Distribute trees;
         Evaluate all trees;
     Find *phylo–clusters*;
     Update distance matrix;
until *All sequences clustered* ;

Figure 3. General scheme of the *Parallel Phyl-Tree* **with simulated annealing.**

4 Experimental validation

We have performed a set of experiments with synthetic as well as real–life biological data to validate our new approach. The experiments were run on a cluster of 15 dual AMD 1400 MHz nodes (one CPU used by the application, one CPU used by OS) connected by FastEthernet network. Each node was equipped with 512MB of RAM and was running under control of Linux. A single node could use 64MB of RAM for *alignment cache* [26] and the persistent cache storage was of infinite capacity (erased before every execution). Our test program as well as ClustalW–MPI (which we utilised for comparison purposes) was compiled using GCC–4.0 compiler. To provide MPI routines we have used mpich2-1.0.2 environment. In the reminder of this paper we use PT when referring to exact search variant of the *PhylTree*, and PT-SA when talking about simulated annealing based version.

4.1 Experiments

In the first experiment a set of 250 DNA sequences of average length 50 bp was generated using the Rose package [25]. The *Relatedness* parameter of the generator was set to 250 and HKY [9] DNA model was used. Such generated sequences were next aligned using both PT and PT-SA method with different scoring functions, and different levels of precision (parameter k). Since SA is a stochastic metaheuristic each execution of PT-SA was repeated 10 times with different seeds of random number generator.

To verify quality of reconstructed alignments and accompanying trees we have measured execution times, sum-of-pairs score (SPS) utilised e.g. in [7], and the mean symmetric distance (TD) between final tree and

Method	Time [s]	SPS	TD
PT SP 4	66	0.338	288
PT SP 5	3326	0.378	316
PT SP 6	9267	0.425	324
PT Pars 4	312	0.139	150
PT Pars 5	783	0.145	140
PT Pars 6	7034	0.167	118
PT-SA SP 5	714 ± 46	0.352 ± 0.002	329
PT-SA SP 6	1312 ± 97	0.366 ± 0.007	348
PT-SA SP 7	7011 ± 257	0.376 ± 0.006	351
PT-SA SP 8	15052 ± 436	0.415 ± 0.017	360
PT-SA Pars 5	246 ± 36	0.143 ± 0.003	189
PT-SA Pars 6	425 ± 30	0.156 ± 0.007	179
PT-SA Pars 7	1614 ± 138	0.171 ± 0.004	180
PT-SA Pars 8	3617 ± 171	0.182 ± 0.003	184
ClustalWMPI	20	0.083	287

Table 1. Result of processing Rose-generated sequences. SPS is sum-of-pairs score (1 means perfect match), DT is symmetric distance (lower values indicate better results).
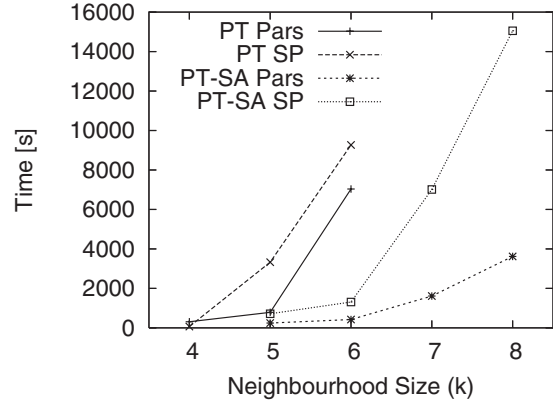

Figure 4. User observed execution times of different variants of *PhylTree* as a function of neighbourhood size k.

Method	Time [s]
PT Pars 4	6015
PT Pars 5	39152
PT-SA Pars 5	20774
PT-SA Pars 6	37246
ClustalW-MPI	678

Table 2. Result of processing of Cyanobacteria rRNA sequences.

the "true" tree [22]. To compute the SPS score the bali_score package was used, and to determine TD distance treedist tool from the Phylip package was utilised.

Table 1 and Figure 4 summarise obtained results. Here, we use SP to denote sum of pairs scoring function, and $Pars$ to denote Sankoff's parsimony criterion. The integer numbers indicate the size of the neighbourhood k utilised during analysis.

To perform the second experiment we have extracted a set of 477 Cyanobacteria small–subunit rRNA sequences from the RDP-II database [3]. The average length of the sequences was 600 bp, with minimal length around 200 bp and maximal 1200 bp. Next, we aligned these sequences using both exact and simulated annealing variants of the *PhylTree*. As a scoring function we have used Sankoff's parsimony criterion. The purpose of this experiment was to measure efficiency of our approach on relatively large set of real sequences. Table 2 presents the obtained results.

4.2 Discussion

As expected, in all cases the user observed execution time of PT-SA is significantly shorter as compared to PT run with the same parameters. When the size of neighbourhood $k > 6$ number of trees which have to be processed by PT is huge, and PT cannot be executed in acceptable time limits. Although results generated by PT-SA are slightly worse than corresponding results

generated by PT (with respect to SPS and TD score), both methods outperform ClustalW–MPI significantly. On the other hand, execution time of ClustalW–MPI is more than 10 times shorter.

When parsimony criterion ($Pars$) is applied, PT-SA with $k = 8$ is faster than PT with $k = 6$. Moreover, it generates alignments of better quality. However, we should notice that this is not in the case of SP function. To process a single neighbourhood of size $k = 8$ PT has to evaluate 135135 trees, while PT-SA evaluates around 1200. Since for each neighbourhood there are several equally parsimonious trees, PT-SA is still able to find topologies close to the optimal. Again, this seems to be not the case for SP function. Here, number of points visited by PT-SA is too small compared to the size of the search space. This results in decreased accuracy of generated alignments and high variance of SPS score (see for example PT-SA with SP and $k = 8$). However, this drawback can be overcame by changing parameters of SA.

The results in Table 1 shows that in both PT and PT-SA cases, the accuracy of generated alignments,

as well as trees, increases for increased values of parameter k. Obviously, it is not the case for the SP function which does not make any assumptions about phylogenetic relations between analysed sequences. On the other hand, application of SP function results in very good SPS score. This can be explained by the fact that SP utilises affine gap penalty which helps to handle long indels generated by Rose.

The Sankoff's parsimony criterion (results indicated with $Pars$) generates better results in the context of tree quality, but these results are still inadequate. This is mainly because because $PhylTree$ works in a progressive manner, thus possible rearrangements of final tree are limited. Moreover, the analysed sequences were short, thus amount of phylogenetic information was limited.

Efficiency of the PT-SA method has been confirmed in the second series of experiments. However, in this case we were not able to assess quality of the generated alignments. The reason is that the reference alignment extracted from the RDPII database is a result of high quality structural comparison with manually aligned seedsequences. As in the first experiment, PT-SA clearly outperforms PT in the terms of running times. The comparison of generated alignments, where output of PT $Pars$ 5 is a reference alignment and PT-SA is tested alignment, resulted with SPS score of 0.92. This may suggest that even for large data sets PT-SA is able to generate results comparable with PT.

5 Conclusions

In this paper we describe a new extension of the Parallel PhylTree method which is based on application of simulated annealing to perform local phylogeny search. This new approach allows execution time of our parallel software to be reduced significantly, while preserving a good quality of the final solutions.

In our current implementation, redundant analysis of neighbourhoods is performed independently. It means that tabu list generated for a given neighbourhood is not taken into account in case when this neighbourhood is processed for the second time. This issue can be solved by storing tabu lists of processed neighbourhoods on the master node for possible reuse. This way efficiency of redundant analysis could be improved.

The source code of the extended Parallel Phyl-Tree can be obtained from the SVN repository at https://lin.icis.pcz.pl/websvn. More detailed results together with accompanying benchmark data will be available for download as well. At present, the Parallel PhylTree with SA search is integrated with our parallel server for multiple sequence alignment, and it can be accessed online via https://hal.icis.pcz.pl/PhyloServer/.

References

[1] S. Altschul, T. Madden, A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped BLAST and PSI–BLAST: a new generation of protein database search programs. *Nuc. Acids Res.*, 25(17):3389–3402, 1997.

[2] D. Barker. LVB: parsimony and simulated annealing in the search fo phylogenetic trees. *Bioinformatics*, 20(2):274–275, 2004.

[3] J. Cole, B. Chai, Q. Wang, S. Chandra, R. Farris, S. Kulam, D. McGarrel, T. Schmidt, G. Garrity, and J. Tiedje. The Ribosomal Database Project (RDP–II): previewing a new autoaligner that allows regular updates and the new prokaryotic taxonomy. *Nuc. Acids Res.*, 31(1):442–443, 2003.

[4] P. Diaconis and S. Holmes. Random walk on trees and matchings. *Elec. J. Prob.*, 7:1–17, 2002.

[5] J. Ebedes and A. Datta. Multiple sequence alignment in parallel on a workstation cluster. *Bioinformatics*, 20(7):1193–1195, 2004.

[6] R. Edgar. MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nuc. Acids Res.*, 32(5):1792–1797, 2004.

[7] P. Gardner, A. Wilm, and S. Washietl. A benchmark of multiple sequence alignment programs upon structural RNAs. *Nuc. Acids Res.*, 33(8):2433–2439, 2005.

[8] F. Glover. Tabu search: A tutorial. *Interfaces*, 20(4):74–94, 1990.

[9] M. Hasegawa, H. Kishino, and T. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *J. Mol. Evol.*, 22(2):160–174, 1985.

[10] D. Higgins, J. Thompson, and T. Gibson. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22:4673–4680, 1994.

[11] D. Hillis, C. Moritz, and B. Mable, editors. *Molecular Systematics*, chapter Phylogenetic Inference. Sinauer Associates, Inc., 1996.

[12] M. Holder and P. Lewis. Phylogeny estimation: Traditional and Bayesian approaches. *Nature Reviews Genetics*, 4(4):275–284, 2003.

[13] T. Jiang, E. Lawler, and L. Wang. Aligning sequences via an evolutionary tree: complexity and approximation. In *Proc. of 26th Ann. ACM Symp. on Theory of Computing*, pages 760–769, 1994.

[14] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[15] J. Kleinjung, N. Douglas, and J. Heringa. Parallelized multiple alignment. *Bioinformatics*, 18(9):1270–1271, 2002.

[16] K. B. Li. ClustalW–MPI: ClustalW analysis using distributed and parallel computing. *Bioinformatics*, 19(12):1585–1586, 2003.

[17] D. Mikhailov, H. Cofer, and R. Gomperts. Performance optimization of ClustalW: Parallel ClustalW, HT Clustal, and MULTICLUSTAL. http://www.sgi.com/industries/sciences/chembio/, 2005.

[18] C. Notredame, L. Holm, and D. Higgins. COFFEE: an objective function for multiple sequence alignments. *Bioinformatics*, 14(5):407–422, 1998.

[19] G. Parmentier, D. Trystram, and J. Zola. Cache-based parallelization of multiple sequence alignment problem. In *Proc. of Euro-Par '04*, pages 1005–1012, 2004.

[20] C. D. Polychronopoulos and D. J. Kuck. Guided self–scheduling: A practical scheduling scheme for parallel supercomputers. *IEEE Trans. on Computers*, 36(12):1425–1439, 1997.

[21] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content–addressable network. In *Proc. of SIGCOMM '01*, pages 161–172, 2001.

[22] D. Robinson and L. Foulds. Comparison of phylogenetic trees. *Math. Biosci.*, 53:131–147, 1981.

[23] M. Schmollinger, K. Nieselt, M. Kaufmann, and B. Morgenstern. DiAlign–P: fast pair–wise and multiple sequence alignment using parallel processors. *BMC Bioinformatics*, 5(1):128, 2004.

[24] A. Stamatakis. An efficient program for phylogenetic inference using simulated annealing. In *Proc. of IPDPS 2005*, 2005.

[25] J. Stoye, D. Evers, and F. Meyer. Rose: generating sequence families. *Bioinformatics*, 14(2):157–163, 1998.

[26] D. Trystram and J. Zola. Parallel multiple sequence alignment with decentralized cache support. In *Proc. of Euro-Par '05*, pages 1217–1226, 2005.

[27] T. Williams, T. Berger–Wolf, B. Moret, U. Roshan, and T. Warnow. The relationship between maximum parsimony scores and phylogenetic tree topologies. Technical Report TR–CS–2004–04, University of New Mexico, 2004.