Ann Oper Res (2011) 183: 75–94
DOI 10.1007/s10479-010-0682-5

Adaptive memory programming: local search parallel algorithms for phylogenetic tree construction

Jacek Blazewicz · Piotr Formanowicz · Pawel Kedziora · Pawel Marciniak · Przemyslaw Taront

Published online: 18 September 2010

Abstract One of the most important aspect of molecular and computational biology is the reconstruction of evolutionary relationships. The area is well explored after decades of intensive research. Despite this fact there remains a need for good and efficient algorithms that are capable of reconstructing the evolutionary relationship in reasonable time.

Since the problem is computationally intractable, exact algorithms are used only for small groups of species. In the Maximum Parsimony approach the time of computation grows so fast when number of sequences increases, that in practice it is possible to find the optimal solution for instances containing about 20 sequences only.

It is this reason that in practical applications, heuristic methods are used. In this paper, parallel adaptive memory programming algorithms based on Maximum Parsimony and some known neighborhood search methods for phylogenetic tree construction are proposed, and the results of computational experiments are presented. The proposed algorithms achieve a superlinear speedup and find solutions of good quality.

Keywords Phylogenetic trees · Parallel algorithms · Local search · Adaptive memory programming · Maximum Parsimony

1 Introduction

Phylogenetic analysis deals with a reconstruction of the evolutionary relationship of taxa (group of species) and is widely used in biological analysis (cf. Kedziora et al. 2005). In most

J. Blazewicz · P. Formanowicz
Institute of Bioorganic Chemistry, Polish Academy of Sciences, Noskowskiego 12/14, 61-714, Poznan, Poland

J. Blazewicz · P. Formanowicz (✉) · P. Kedziora · P. Marciniak · P. Taront
Institute of Computing Science, Poznan University of Technology, Piotrowo 3a, 60-965 Poznan, Poland
e-mail: piotr@cs.put.poznan.pl

P. Kedziora
e-mail: pawel.kedziora@cs.put.poznan.pl

cases the cause of the evolution are random mutations in genomes. Most of such mutations have no immediate effect on the organisms, or they are lethal for them (if such a mutation alters the function of some important gene). However, some of them cause the genes to gain new functionality. It may happen that an organism carrying such a modified gene becomes better adapted to its environment. In such a situation the mutant has a greater chance of survival and replication. This means that the mutation becomes established.

The evolutionary relationship of species is usually shown as a phylogenetic tree. In such a tree the root represents an ancestor of all species represented by other nodes of the tree. Leaves are the species living at present while the internal nodes of the tree represent some hypothetical species that appeared during the course of evolution. A phylogenetic tree is only a hypothesis about evolution. It is important to remember that it always shows only one of the possible ways of the evolution.

Building a phylogenetic tree for a group of organisms is not easy, even for a group of moderate size. There are at least two sources of difficulties. First, algorithms for phylogenetic tree construction are based on estimated models of the evolution. This is also the reason for which an evolutionary tree is only a hypothesis. Second, for a construction of such a tree only data concerning the species living at present are available, i.e. those that correspond to the leaves in the tree. It means that the tree is reconstructed on the basis of the similarity among data representing the species living at present.

There are two main classes of methods for building phylogenetic trees (cf. Gusfield 1997; Setubal and Meidanis 1997), i.e. the ones based on similarities among the features (or characters) that describe a species (character based methods), and methods based on the evolutionary distances between species (distance based methods). In this paper we will focus only on the methods of the first class.

The data used by the methods of the first class are usually sequences of nucleotides or amino acids. Nucleotides are molecules that comprise the structural units of RNA and DNA. The vast majority of living organisms encode their genetic information in long strands of DNA (the only exceptions are some viruses that use RNA instead of DNA). These units are of four types denoted by A, T, G and C, and the genetic information is encoded in the DNA sequence. The DNA molecules are copied and inherited across generations. Traits, which are features of organisms, are encoded in DNA as instructions for constructing and operating an organism. The instructions are contained in segments of DNA called genes. A gene is a basic unit of heredity in a living organism and its function is to provide information necessary for some molecular mechanisms to build proteins that are basic blocks of tissues. First, a DNA fragment encoding a gene is copied into a similar molecule called mRNA (this process is called transcription). mRNA is a template for a structure called a ribosome, which translates the sequence of nucleotides into a corresponding sequence of amino acids that compose a protein (this process is called translation).

In the case of character based methods, a feature is a position in the nucleotide or amino acid sequence and a nucleotide or amino acid present at this position is a value (a state) of the feature. Similarities occurring between subsequences of given fragments of nucleic acids or proteins are used as a basis for the tree construction, which is built in such a way that the total number of state changes on all the paths from the root to any leaf is minimal. Such a tree is called to be the *most parsimonious one* (cf. Edwards and Cavalli-Sforza 1963).

Many algorithms for phylogenetic tree construction are known. However, since the problem of such a construction is generally computationally intractable (cf. Day et al. 1986; Foulds and Graham 1982; Setubal and Meidanis 1997), polynomial-time exact algorithms are known only for special cases. Such algorithms are useful in a limited number of cases. The Maximum Parsimony Problem is known to be NP-Hard (cf. Foulds and Graham 1982).

In general, an exhaustive search may involve $\frac{(2n-5)!}{(n-3)!2n^3}$ (unrooted, bifurcating tree topologies) or $\frac{(2n-3)!}{(n-2)!2n^2}$ (rooted, bifurcating tree topologies) possible solutions for a case with n taxa involved (cf. Edwards and Cavalli-Sforza 1964). In general, the branch and bound algorithms (cf. Hendy and Penny 1982; Blazewicz et al. 2004), which finds the optimal solution each time it is applied, is not efficient enough for handling large size instances. The size of instance in this case is limited to about 20. Hence, heuristic methods are proposed that are able to construct trees based on more general models (cf. Andreatta and Ribeiro 2005; Goëffon et al. 2005; Lin et al. 2007; Ribeiro and Vianna 2005). Another way to manage the intrinsic intractability of the problem is to design parallel algorithms (cf. Blazewicz et al. 2004; Stamatakis 2004), which are able to expand the size of instances solved in practice. The last approach is especially promising, since parallel computers and large clusters are increasingly commonly.

The main goal of this work is not to design a new method for constructing phylogenetic trees, but rather to develop a strategy of cooperation of distributed computational nodes using an adaptive memory heuristics. In order to facilitate the considerations, we have not used the well known, efficient, but complex methods of finding a phylogenetic tree such as: parsimony ratchet (cf. Nixon 1999), TNT (cf. Goloboff 1999) or Recursive-Iterative DCM3 (cf. Roshan et al. 2004). Instead, a local search procedure with adaptive memory feature was used. However, we believe that the proposed methods could be successfully applied to the above mentioned algorithms of finding phylogenetic trees. Obviously, the developed method should lead to finding phylogenetic trees of good quality in a reasonable time.

In this paper, parallel adaptive memory programming (AMP) algorithms for parsimonious phylogenetic tree construction are proposed. In order to eliminate the influence of a style of implementation of algorithms on efficiency of applications, a general framework is designed. This work provides an architectural basis for comparison. Frameworks offer the architecture patterns and solutions to develop a family of similar systems in the same application domain (cf. Andreatta and Ribeiro 2005; Fayad and Schmidt 1997).

The term adaptive memory programming was introduced by Glover and was originally related to taboo search (cf. Glover 1989, 1997). Afterwards the term was extended by Taillard et al. (2001) to all algorithms that are coherent with the following scheme:

```
1. Initialize the memory.
2. While a stopping criterion is not met do:
   2.1 Generate a new provisional solution $s$ using data stored in the memory.
   2.2 Improve $s$ by a local search; let $s'$ be the improved solution.
   2.3 Update the memory using the pieces of knowledge brought by $s'$.
```

As one can notice, the algorithms presented in Sect. 3 satisfy the above scheme.

Three sets of experiments were performed to test the proposed algorithms. In the first set of experiments, the speedup of parallel algorithms was measured on real instances, which usually are especially hard from the computational point of view. In the second, the quality of the obtained solutions was measured and tested against solutions obtained by other known methods. Here instances (besides the first instance) were generated with an artificial evolution tool Rose (cf. Stoye et al. 1998). Both tests showed good (in some cases superlinear) speedup of the algorithms and good quality of the obtained results outperforming those obtained by the existing software packages. In the third set of experiments, the comparison with other known methods was performed on real-life benchmark instances.

The organization of the paper is as follows. In Sect. 2, the sequential versions of the methods are described. In Sect. 3, the parallel algorithms are presented, and in Sect. 4, the results of the computational experiments are shown. The paper ends with a conclusion in Sect. 5.

2 The methods of neighborhood search

The methods whose parallel versions are proposed in this paper are character-based parsimony algorithms. A set of sequences representing the analyzed species is used as input to the algorithms. The goal is to find a phylogenetic tree that explains the data best, i.e. minimizes objective function f. The algorithms considered in this paper belong to the class of neighborhood search methods. Algorithms of this type check trees that can be obtained from a current solution by changing it in some specified way, i.e. they check the neighborhood of this solution. For each of the elements of the neighborhood, the value of the objective function f is calculated and the tree with the best value is chosen as the next solution. In the character based methods, the objective function f should be chosen in such a way that the best tree will explain the observed diversity in the input data with a minimal number of character values substitutions.

The values of the objective function f used in this work can be calculated using Fitch's algorithm in the following way (cf. Fitch 1971):

The input to the algorithm is the topology of a phylogenetic tree with n nodes and set A of possible character values, where $|A| = k$. The algorithm assigns the values of a single character (let us note that here a character is equal to a position in a sequence labeling a node) for every node in the tree.

Let us denote the value of the selected character for node v by c_v.

Step 1. For each node v of the tree assign set $S_v \subseteq A$ in the following way.
If v is a leaf, then $S_v = \{c_v\}$. If v is an internal node whose immediate successors are nodes u and w, $S_v = S_u \cap S_w$ if $S_u \cap S_w \neq \emptyset$, and $S_v = S_u \cup S_w$ otherwise. In order to determine set S_v for all the nodes of the tree, it is traversed in the post-order way starting with leaves and working upwards.

Step 2. Having determined set S_v for every node v of the tree, a particular value c_v from the set is assigned to each internal node v in the following way. The tree is traversed in preorder, i.e. from the root downwards. If for immediate predecessor u of node v, $c_u \in S_v$, the algorithm sets $c_v = c_u$. Otherwise, randomly chosen $t \in S_v$ is assigned to c_v (which applies also to the root).

Let us note that the above procedure must be repeated for every position (character) in the labels assigned to the nodes. As a result, a fully labeled tree is obtained after the algorithm runs for every character. The number of changes, being the value of the objective function f (in what follows we will refer to the objective function f as to the Fitch score), is equal to the number of times the union of character values is created in Step 1.

Searching through selected tree topologies is carried out with a local search procedure that is described in the following way:

1. Choose some feasible solution T.
2. Generate neighborhood $N(T)$ of T.
3. If there is no $T' \in N(T)$ such that $f(T') - f(T) < 0$ then STOP.
4. Choose some $T' \in N(T)$ for which $f(T') - f(T) < 0$.
5. Set $T = T'$.
6. Go to Step 2.

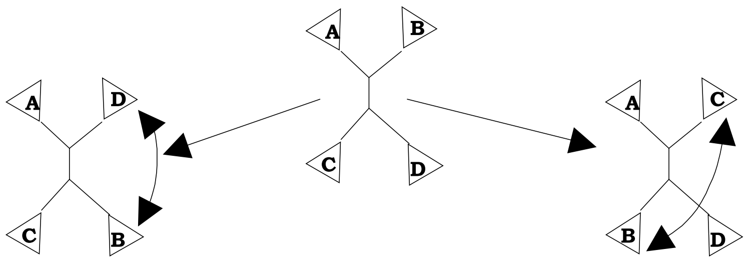

Fig. 1 An example of Nearest Neighborhood Interchanges (NNI). A, B, C and D are subtrees. Subtree B is swapped with subtrees D and C

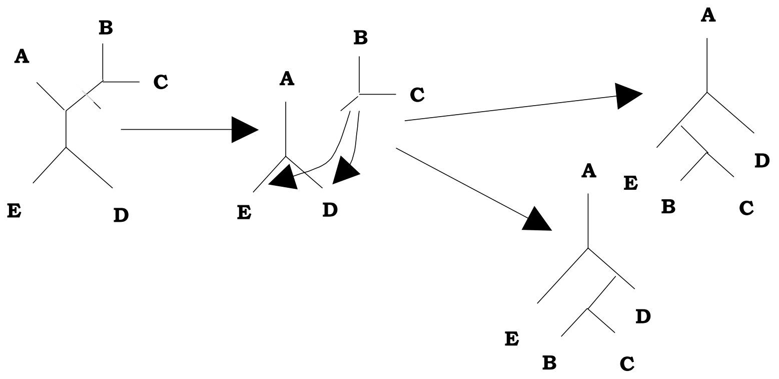

Fig. 2 An example of Subtree Pruning and Regrafting (SPR). A, B, C, D and E are species. The rooted subtree which contains species B and C is removed from the tree and then connected using the same root to any edge in the tree containing species A, D and E

The above procedure is only a general framework that can be modified in many ways. Possible variants are obtained by specifying a neighborhood generation method, a way of choosing the solution for the next iteration of the procedure (e.g. First Improvement Move (FIM), Best Improvement Move (BIM)) and a method of generating the starting solution.

In the parsimony problem, a neighborhood of a given solution T is a set of trees obtainable from T by some elementary changes in the tree topology. The most common strategies for changing the topology are Nearest Neighborhood Interchanges (NNI), Subtree Pruning and Regrafting (SPR), and Tree Bisection and Reconnection (TBR) (cf. Felsenstein 2004).

For the NNI strategy, the neighborhood is obtained by swapping subtrees attached to some edge (cf. Fig. 1).
In the SPR strategy the elements of the neighborhood of T are constructed by removing some edge from T that results in a creation of two trees t and $T - t$. The root of tree t can be later connected (regrafted) to any edge of $T - t$ (cf. Fig. 2).
The TBR strategy is similar to the SPR except that after a bisection of tree T, subtree t can be reconnected to $T - t$ by some new edge connecting an arbitrarily chosen edge of t with any edge of $T - t$ (cf. Fig. 3). Note that in the SPR method the connection point of t has to be the root.

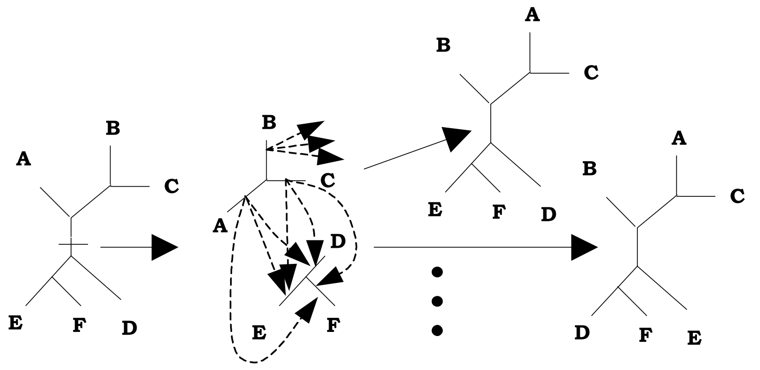

Fig. 3 An example of Tree Bisection and Reconnection (TBR). A, B, C, D, E and F are species. Bisection divides the whole tree T into subtree t representing organisms A, B and C and subtree $T - t$ representing species D, E and F. The reconnection can be made by an addition of a new edge which joins arbitrary edges from t and $T - t$

3 Parallelization of the methods

The presented approach is based on the character data and the neighborhood generating strategies described in the previous section. The solution space is searched by the local search procedure with an adaptive memory mechanism. All of the proposed parallel algorithms are based on the same framework; this way the influence of implementation details on the obtained results is minimized (cf. Andreatta and Ribeiro 2005; Fayad and Schmidt 1997). The framework includes classes responsible for communication, generation of starting random solutions and starting solutions obtained by a decomposition of local optima, construction of the neighborhood, etc.

The only elements of the algorithms by which they differ are strategies of generating the starting points of the local search procedure. The starting points are created based on local optima already visited.

Three versions of the parallel local search algorithms were created (they are described momentarily). Each of the versions was combined with three neighborhood generation strategies (i.e. NNI, SPR, and TBR) as well as with a Variable Neighborhood Search (VNS) method (cf. Mladenović and Hansen 1997). The last method is based on a simple principle: systematic change of neighborhood to a wider neighborhood when a local optimum is found, in order to counteract getting stuck in a local optimum. (The proposed algorithms use VNS as a combination of the NNI, SPR, and TBR methods. The algorithms try to find a local optimum using the NNI. When found, such an optimum becomes a starting point for the local search procedure combined with the SPR. Next, the solution becomes a starting point for the local search procedure combined with the TBR.) The combinations of the local search procedures with neighborhood generation strategies result in twelve versions of the algorithms. Each of them can be further modified by combining with different methods of choosing a solution for the next iteration, i.e. the FIM or the BIM.

The parallel algorithms have the master-slave structure. At the beginning of computations, the master process generates 30 starting solutions (i.e. phylogenetic trees), which are then sent on the request to slave processes. A starting tree is generated by adding to the tree at random points of the tree (based on a uniform distribution) species from the set of species not yet added to the tree. A species is chosen from the set randomly also based on a uniform distribution.

Each of the slaves takes the tree obtained from the master as a starting point for the local search procedure. When a local optimum has been found by a slave, it is sent to the master process, which collects the local optima that are used to create new starting solutions for slave processes. The motivation for such a strategy is the hypothesis that local optima can contain fragments of the global optimum. These fragments are subtrees of local optima, and hence the master builds a new starting solution using "good" subtrees of locally optimal trees. A subtree is "good" if it has not been changed in a number of iterations.

We have assumed that a subtree remains unchanged during the local search procedure when it is hard to find a better subtree or if such a subtree does not exist. In order to identify such subtrees the number of iterations during which the structure of the subtree rooted in a given node has not been changed, is stored.

3.1 Parallel version #1 (PV1)

In this version of the algorithm the master stores the local optima received from slaves in a FIFO queue. When a slave requests a new starting point, the master selects the first tree from the queue and extracts from it all subtrees for which the number of iterations without structure change is greater than some threshold. These subtrees are then connected in order to create a new complete tree for the considered set of species. If the tree is incomplete (which is the most common case) the remaining internal nodes are added to the tree in a random way. If the queue is empty, the master creates a fully random tree. The method of adding nodes to the tree or creating a fully random tree is the same as the method of generation of starting solutions.

The new starting solution is sent to the requesting slave and the tree used for its construction is removed from the queue.

3.2 Parallel version #2 (PV2)

In this version the way of collecting local optima is analogous to the one used in version #1. The new starting point for a slave process is assembled by the master from subtrees of the trees stored in the FIFO queue. The tree from the top of the queue is taken and the biggest stable subtree is selected. Like in version #1 a subtree is considered to be stable when the number of iterations without change of its structure is greater than some threshold. This subtree becomes a part of a new starting solution. Next, the second tree from the queue is taken and its biggest stable subtree is added to the constructed new tree. Here, if some group of species in the subtree that is to be added to the new starting point already exists in the partially constructed new tree, then this group is removed from the subtree. The remaining part of the subtree (the subtree that is to be added to the new starting point) is added to the newly created starting solution. This procedure is continued until a complete tree is created or until there are no trees in the queue. In the second case, the remaining species are randomly joined to a new tree. When the new starting point is sent to the slave the tree from the top of the queue is removed.

3.3 Parallel version #3 (PV3)

In this version every node in the locally optimal tree received by the master is labeled by a value of the product of the number of iterations without subtree (rooted in this node) structure change and the number of leaves in this subtree. This value measures subtree "goodness". Roughly speaking, a subtree is "good" if it is big and stable. It should be noted that the number of iterations without subtree structure change decreases when the level of a subtree root decreases. The explanation of this fact is very simple. Subtrees that are closer to the root of a tree contain more internal nodes than subtrees that are more distant. Hence, it is more probable that after a given number of iterations a bigger subtree (with a higher level of its root) has changed than a subtree with a lower level of its root. The measure defined above is a compromise between high stability (which is the highest for subtrees composed of only two nodes) and the size of the subtree.

In this version of the algorithm, subtrees with the greatest value of the measure are stored in the queue (instead of the whole trees, as in versions #1 and #2). The elements of the queue are ordered according to the value of the measure. From each locally optimal tree, subtrees which cover at least 60% of species in total are extracted. However, if the subtree with greatest value of the measure is composed of only one species, the extraction is stopped.

The ordering of the extraction of the subtrees is determined by the measure. A new starting solution is created from the subtrees having the greatest value of the measure, i.e. being at the top of the queue. The missing nodes (nodes that are not present in the subtree) are added to the new tree in random places of the tree structure. At the end, the best subtree is removed from the queue.

Before the insertion of any subtree into the queue, the algorithm checks whether the subtree has been already used to construct a starting tree. Since comparing the subtree with the list of all subtrees used so far would be time consuming, the algorithm checks only if a subtree with the same Fitch score (the score is calculated using Fitch's algorithm presented in Sect. 2) and with the same group of species has not been used so far. A special list is used to perform this checking. The elements of the list are sets of species composed from subtrees, and the corresponding Fitch scores.

The value of a subtree measure is decreased by some factor if from the above mentioned checking it follows, that a subtree with the same Fitch score and the same group of species has already been used to construct a starting tree. The factor increases every time such a subtree is going to be inserted into the queue. This mechanism should prevent the method from falling into a loop where the new starting points are constructed from a few locally optimal trees (which leads the slaves to find the same trees many times and put their subtrees at the top of the queue).

4 Computational experiments

The algorithms described in the paper were implemented in C++ using the MPI library and tested on the Sun Fire 6800 machine (UltraSPARC III+ 900 MHz processor) of the Poznan Supercomputing and Networking Center. Several types of experiments were performed. In the first set of experiments, the speedup of parallel algorithms was measured on real instance, especially hard from the computational perspective. In the second set of experiments, the quality of the obtained solutions was measured and tested against solutions obtained by other known methods. Here, the instances (besides the first instance) were generated with the use of an artificial evolution tool Rose (cf. Stoye et al. 1998). In the third set of experiments, the comparison with other known methods based on Maximum Parsimony local search procedures was performed. We used real-life benchmark instances taken from (Ribeiro and Vianna 2005), which were initially presented in Luckow et al. (1985).

The first and the second sets of tests were carried out on three instances. The first one, denoted as TI-1, was composed of 100 real sequences encoding part of the E1 protein of

HCV virus (in fact, the sequences belong to different mutations of this virus). The sequences for this instance were taken from the HCV database located at the following address: http://s2as02.genes.nig.ac.jp/. The length of these sequences was equal to 465 nucleotides. The Fitch score of the best known solution for this instance equals 2644.

The second and the third instances, denoted as TI-2 and TI-3, were composed of 100 artificial sequences created by the "Rose—random model of sequence evolution" tool (cf. Stoye et al. 1998), which provides a probabilistic model of the evolution of RNA-, DNA-, or protein-like sequences. Rose is commonly used, and its results are useful for the evaluation of methods for multiple sequence alignment construction and the prediction of phylogenetic relationships (cf. Nelesen et al. 2008). During this artificial evolutionary process, the 'true' history is logged and the 'correct' multiple sequence alignment is created simultaneously. Hence, on this basis we could compare resulting trees generated by our methods with the trees that were artificially evolved. The lengths of these sequences were equal to 500 and 1000 nucleotides, respectively. The Fitch score of the best known solutions for these instances as well as the score for the trees originally generated by Rose are equal to 7905 and 15844, respectively.

Before starting the tests, the sequences composing each of the instance were preprocessed to remove characters in the sequences whose impact on the score (i.e. the value of the objective function computed by Fitch's algorithm) could be determined on the basis of other positions. The preprocessing involves selecting all pairs of columns such that some bijection is satisfied between these columns. For each pair for which the bijection holds, an arbitrary column from the pair is removed from the input sequences and a weight of the remaining column is increased by the weight of the removed column. The Fitch procedure calculates the score for the remaining column and afterwards multiplies the calculated score by its weight. The length of the sequences was equal to 319 nucleotides for the first instance after pre-processing. This process did not alter the second (TI-2) and the third (TI-3) instances.

We considered two stopping criteria. The first was a value of the solution obtained while the second was based on a limit on computation time. According to the latter all processes stopped the computation after a given period of time.

In the first set of experiments, computation times of the parallel algorithms for instance TI-1 were measured. Each of the parallel versions was combined with the NNI local search procedure and the FIM method of choosing the solution for the next iteration. The computations were stopped and the time was measured if a solution with the Fitch score not greater than 3400 was found. The value 3400 is about 30% worse than the best known solution for that instance, so it seems to be a good stopping point for the tested algorithms. The results are shown in Table 1. Because the result of each execution of a given algorithm can be different (due to heuristic strategy and a nondeterministic behavior of the parallel environment),

Table 1 Computation time in seconds of the parallel algorithms for instance TI-1

Method	No. of processors							
	1		4		8		16	
	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.
PV0	4905.31	5646.48	1578.76	1224.40	768.50	649.54	342.02	357.96
PV1	659.19	257.41	147.33	55.43	69.68	34.38	51.42	24.88
PV2	1300.41	1178.79	178.69	91.02	109.81	71.63	63.76	25.37
PV3	6395.28	6495.89	759.13	1027.61	570.93	705.75	182.55	195.88

Table 2 Speedup of the algorithms obtained for instance TI-1 where the reference computation time of sequential algorithm is a given algorithm executed on one processor

Method	No. of processors			
	1	4	8	16
PV0	1.0	3.1	6.4	14.3
PV1	1.0	4.5	9.5	12.8
PV2	1.0	7.3	11.8	20.4
PV3	1.0	8.4	11.2	35.0

Table 3 Speedup of the algorithms obtained for instance TI-1 where the reference computation time of sequential algorithm is PV0 algorithm executed on one processor

Method	No. of processors			
	1	4	8	16
PV0	1.0	3.1	6.4	14.3
PV1	7.4	33.3	70.4	95.4
PV2	3.8	27.5	44.7	76.9
PV3	0.8	6.5	8.6	26.9

each entry contains an average over 20 runs. The abbreviations: PV1, PV2 and PV3 denote parallel versions #1, #2 and #3, respectively. PV0 means that new starting points were generated randomly. This row shows the impact of the applied adaptive memory mechanisms and the parallelization strategy on the algorithms' efficiency.

Tables 2 and 3 present the speedup of the parallel versions of the algorithms. The speedup presented in Table 2 is a ratio of the computation time of a given algorithm and the computation time of this algorithm run on one processor. Table 3 presents the speedup calculated as a ratio of the computation time of a given algorithm and the computation time of a sequential algorithm without the adaptive memory (PV0) run on one processor. Analyzing Table 2 one notices that versions PV2 and PV3 have bigger speedups than the number of processors. Furthermore, from Table 3 it follows that all versions have superlinear speedup, which shows that the cooperation of processors and adaptive memory programming has an influence on the computation time and the obtained results. Table 3 shows that the proposed mechanisms used in versions PV1 and PV2, even for sequential algorithms (see results in the column where the number of processors equals 1), gives much better results than the algorithm where a starting point for a local search procedure is generated randomly (PV0).

In the second experiment, results obtained by the parallel algorithms combined with various local search procedures were compared. The FIM method of neighborhood generation was used. Here the sequences comprising all three instances were used. The Fitch scores of the trees found within a time limit equal to 600 seconds applied to the whole parallel system are shown in Table 4 (for instance TI-1), where again each entry contains an average over 20 runs. VNS1 is a variant of VNS method where the NNI and the SPR local search procedures are used, while in VNS2, the NNI and the TBR are applied.

In the test whose results were presented in Tables 1 and 4 the FIM method for choosing a solution for the next iteration was applied. The BIM method was not used because of its poor time efficiency and solution quality similar to the one of the FIM. A comparison of the computation time necessary for finding a local optimum by various local search algorithms combined with the BIM and the FIM, respectively, is shown in Table 5. In Table 6 the Fitch scores of the local optima found by the local search methods are presented.

Table 4 The Fitch scores of the trees found by the algorithms in the time limit of 600 seconds for instance TI-1

Method	No. of processors							
	1		4		8		16	
	Score	Std. dev.	Score	Std. dev.	Score	Std. dev.	Score	Std. dev.
PV0+NNI	3573.90	125.23	3470.00	117.56	3356.15	119.41	3342.65	132.81
PV0+VNS1	2653.67	4.06	2650.25	2.12	2648.80	1.36	2648.20	1.44
PV0+VNS2	2651.05	2.58	2648.80	1.51	2648.60	1.19	2647.95	1.10
PV0+TBR	2651.40	2.21	2649.20	1.51	2649.15	1.39	2647.80	1.15
PV1+NNI	3410.80	153.35	3080.50	124.70	2995.10	111.20	2963.50	89.31
PV1+VNS1	2652.16	3.89	2651.10	2.29	2649.10	1.41	2648.25	0.97
PV1+VNS2	2652.95	2.87	2649.30	1.87	2648.90	1.12	2647.10	0.97
PV1+TBR	2651.20	3.40	2649.00	1.41	2648.75	0.91	2647.45	0.76
PV2+NNI	3461.30	183.66	3086.70	160.91	2947.00	135.62	2795.70	60.58
PV2+VNS1	2653.00	3.88	2650.70	1.95	2649.15	1.79	2648.20	1.28
PV2+VNS2	2651.60	1.96	2649.40	1.60	2648.15	1.27	2646.65	0.59
PV2+TBR	2652.20	2.65	2649.90	1.55	2648.10	1.80	2646.65	0.93
PV3+NNI	3588.50	172.76	3418.05	150.47	3354.65	147.24	3272.25	141.12
PV3+VNS1	2653.94	4.89	2649.75	1.86	2649.15	1.18	2648.60	1.05
PV3+VNS2	2651.80	1.88	2649.05	1.19	2648.10	1.25	2647.65	1.14
PV3+TBR	2652.40	2.58	2649.95	0.89	2648.70	1.38	2647.10	1.02

Table 5 Computation time in seconds for local search procedures for instance TI-1

BIM/FIM	Local search procedure					
	NNI		SPR		TBR	
	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.
FIM	6.05	2.03	2698.82	917.80	205.00	38.97
BIM	5.55	0.91	1415.67	108.14	8745.57	549.91
	VNS1		VNS		VNS2	
	Time	Std. dev.	Time	Std. dev.	Time	Std. dev.
FIM	341.69	121.22	363.31	125.47	145.69	27.72
BIM	975.22	126.11	1143.05	115.69	6099.24	333.88

In both cases, the searching was started from a randomly generated tree. As previously, each entry provides an average for 20 runs of an algorithm. The results presented in these two tables justify using only the FIM method in the other tests.

The tests, whose results are presented in Table 1, showed that the fastest version of the parallel algorithm is PV1, but the difference between computation times of PV1 and PV2 decreases if the number of processors increases.

Moreover, the NNI local search procedure finds poor solutions in comparison to all other tested methods, as follows clearly from Table 4. On the other hand, this method quickly

Table 6 The Fitch scores of trees found by local search procedures for instance TI-1

BIM/FIM	Local search procedure					
	NNI		SPR		TBR	
	Score	Std. dev.	Score	Std. dev.	Score	Std. dev.
FIM	4470.95	241.00	2652.20	3.25	2654.20	3.55
BIM	5279.60	250.64	2658.80	5.36	2654.65	3.77
	VNS1		VNS		VNS2	
	Score	Std. dev.	Score	Std. dev.	Score	Std. dev.
FIM	2654.60	5.33	2654.55	5.39	2656.45	4.22
BIM	2656.40	3.91	2655.55	5.20	2655.20	3.62

Table 7 P-values of Mann-Whitney-Wilcoxon test applied to the series of Fitch scores presented in Table 4 (obtained using 16 processors within the time limit of 600 seconds)

	PV0+NNI	PV0+VNS1	PV0+VNS2	PV0+TBR	PV1+NNI	PV1+VNS1	PV1+VNS2	PV1+TBR	PV2+NNI	PV2+VNS1	PV2+VNS2	PV2+TBR	PV3+NNI	PV3+VNS1	PV3+VNS2	PV3+TBR
PV0+NNI	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.00	0.00	0.00
PV0+VNS1		1.00	0.63	0.39	0.00	1.00	0.01	0.05	0.00	0.80	0.00	0.00	0.00	0.27	0.21	0.01
PV0+VNS2			1.00	0.78	0.00	0.41	0.03	0.17	0.00	0.69	0.00	0.00	0.00	0.07	0.41	0.03
PV0+TBR				1.00	0.00	0.23	0.05	0.28	0.00	0.50	0.00	0.00	0.00	0.03	0.60	0.06
PV1+NNI					1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
PV1+VNS1						1.00	0.00	0.01	0.00	0.62	0.00	0.00	0.00	0.16	0.09	0.00
PV1+VNS2							1.00	0.23	0.00	0.01	0.04	0.10	0.00	0.00	0.18	1.00
PV1+TBR								1.00	0.00	0.08	0.00	0.01	0.00	0.00	0.67	0.28
PV2+NNI									1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
PV2+VNS1										1.00	0.00	0.00	0.00	0.11	0.23	0.01
PV2+VNS2											1.00	0.82	0.00	0.00	0.00	0.10
PV2+TBR												1.00	0.00	0.00	0.01	0.18
PV3+NNI													1.00	0.00	0.00	0.00
PV3+VNS1														1.00	0.01	0.00
PV3+VNS2															1.00	0.17
PV3+TBR																1.00

finds local optima (but, as we mentioned, they are rather poor). It should be also noticed that increasing the number of processors has the greatest impact on the quality of the results provided by the NNI method, while the quality of the solutions found by the other methods is almost independent of the number of processors used. In addition, all these methods (except for the NNI) found trees with the Fitch scores being in a very narrow range and very close to the best known. The standard deviation for these methods is low.

As can be noticed, the methods where new starting points are constructed on the basis of already found local optima (PV1-3) provide better or comparable results than the methods

Table 8 Statistical significance of differences between various version of the algorithms. The tests were applied to the series of Fitch scores presented in Table 4 (obtained using 16 processors within the time limit of 600 seconds)

	PV0+NNI	PV0+VNS1	PV0+VNS2	PV0+TBR	PV1+NNI	PV1+VNS1	PV1+VNS2	PV1+TBR	PV2+NNI	PV2+VNS1	PV2+VNS2	PV2+TBR	PV3+NNI	PV3+VNS1	PV3+VNS2	PV3+TBR
PV0+NNI	–	+	+	+	+	+	+	+	+	+	+	+	–	+	+	+
PV0+VNS1		–	–	–	+	–	+	–	+	–	+	+	+	–	–	+
PV0+VNS2			–	–	+	–	+	–	+	–	+	+	+	–	–	+
PV0+TBR				–	+	–	+	–	+	–	+	+	+	+	–	–
PV1+NNI					–	+	+	+	+	+	+	+	+	+	+	+
PV1+VNS1						–	+	+	+	–	+	+	+	–	–	+
PV1+VNS2							–	–	+	+	+	–	+	+	–	–
PV1+TBR								–	+	–	+	+	+	+	–	–
PV2+NNI									–	+	+	+	+	+	+	+
PV2+VNS1										–	+	+	+	–	–	+
PV2+VNS2											–	–	+	+	+	–
PV2+TBR												–	+	+	+	–
PV3+NNI													–	+	+	+
PV3+VNS1														–	+	+
PV3+VNS2															–	–
PV3+TBR																–

where the starting points are generated randomly (PV0), while the computation times are noticeable lower. It is important to verify if the observed differences are significant. To check whether differences between the results produced by the proposed methods are statistically significant we used the Mann-Whitney-Wilcoxon test. We used this test because the result of the Shapiro-Wilk test was negative. The test was applied to the series of the Fitch scores presented in Table 4 which was executed on 16 processors. Table 7 presents the obtained p-values. Table 8 shows (using the "+" symbol) the statistically significant differences between the results produced by the methods, when the significance level was 0.05. The tests were performed in the R environment for statistical computing (cf. Development Core Team 2008). The tests have shown that in most cases the differences between the results provided by the various variants of the proposed methods are statistically significant. However, it can be observed that in the case of PV0 algorithm the influence of the various neighborhood generation strategies on the obtained results is insignificant (except NNI which gives very poor solutions). On the other hand, the most significant are the differences between the results generated using PV2 and the rest of the methods.

In order to estimate distance from an optimum solution, we used the artificially generated instances, i.e. TI-2 and TI-3. Tables 9 and 10 present the Fitch scores of the trees found within a time limit. In the first of these tables the limit is equal to 300 seconds and is applied to instance TI-2 while in the second one the limit is equal to 500 seconds and is applied to instance TI-3.

In almost all cases, the algorithms in the given time limit reached the best known solution. Only the versions based on the TBR could not find satisfactory solutions. This is probably

Table 9 The Fitch scores of the trees found by the algorithms in the time limit of 300 seconds for instance TI-2

Method	No. of processors							
	1		4		8		16	
	Score	Std. dev.	Score	Std. dev.	Score	Std. dev.	Score	Std. dev.
PV0+NNI	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV0+VNS1	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV0+VNS2	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV0+TBR	16025.50	2857.10	15851.50	2796.97	14984.60	3732.46	13135.00	4501.74
PV1+NNI	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV1+VNS1	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV1+VNS2	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV1+TBR	17000.70	219.84	16794.30	120.47	16759.90	110.58	16635.30	131.30
PV2+NNI	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV2+VNS1	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV2+VNS2	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV2+TBR	15108.20	3800.90	16873.00	69.66	15043.00	3762.97	13168.80	4530.85
PV3+NNI	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV3+VNS1	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV3+VNS2	7905.00	0.00	7905.00	0.00	7905.00	0.00	7905.00	0.00
PV3+TBR	15229.60	3862.46	16812.60	94.30	16745.60	59.78	14931.70	3703.86

Table 10 The Fitch scores of the trees found by the algorithms in the time limit of 500 seconds for instance TI-3

Method	No. of processors							
	1		4		8		16	
	Score	Std. dev.	Score	Std. dev.	Score	Std. dev.	Score	Std. dev.
PV0+NNI	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV0+VNS1	17616.30	5604.50	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV0+VNS2	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV0+TBR	28017.30	6809.85	24902.20	6256.37	19317.10	1156.19	17924.90	1233.79
PV1+NNI	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV1+VNS1	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV1+VNS2	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV1+TBR	31622.70	5397.27	22778.80	4050.11	19407.30	1321.66	18592.20	2081.94
PV2+NNI	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV2+VNS1	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV2+VNS2	19473.00	7654.02	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV2+TBR	28231.40	8359.21	22232.50	6407.21	20326.80	1861.12	17274.50	998.22
PV3+NNI	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV3+VNS1	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV3+VNS2	15844.00	0.00	15844.00	0.00	15844.00	0.00	15844.00	0.00
PV3+TBR	24221.30	6915.56	22905.00	4782.39	18909.10	2383.01	18225.50	1614.08

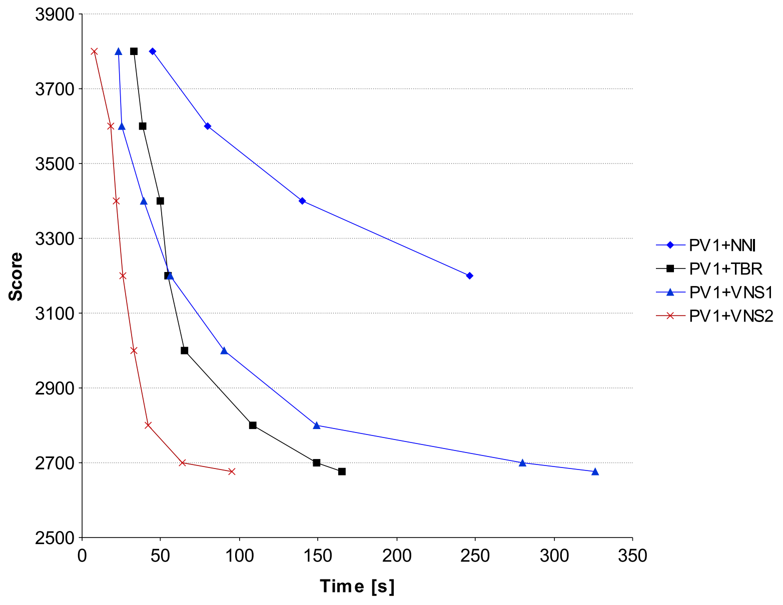

Fig. 4 A relation of the Fitch score for instance TI-1 and the time of the algorithm's execution

caused by the huge neighborhood generated by the TBR, and because of this fact the method could not perform sufficient number of steps of local search.

In the successive computational experiments, whose results are presented in Table 11, the comparison of our algorithms with related methods published in the literature, was performed. Here we used eight real-life test instances (ran on 16 processors) taken from (Ribeiro and Vianna 2005), which were initially presented in Luckow et al. (1985). Columns "Score" and "Time" show the results obtained using selected versions of our proposed algorithms. Subcolumns "Avg.", "Std. dev." and "Best" of "Score" column show average score, its standard deviation and the best solution found, respectively. Subcolumns "Avg." and "Std. dev." of "Time" column show average time in seconds and its standard deviation, respectively. Each entry in these columns has been calculated on the basis of at least 10 runs of a given algorithm and a given instance.

In the next columns we present results of computational experiments published in the literature for these test instances. The column Grasp+VND shows the best values found by Grasp+VND procedure (cf. Ribeiro and Vianna 2005) in subcolumn "Best" and the average computation time in seconds in subcolumn "Time", measured on 2 GHz Pentium IV processor. The authors did not published the average scores. One observes that in most cases our methods also reached the best known solutions, however in noticeable lower average times.

The PMN column shows results, obtained by parallel multi-neighborhood local search heuristic, published by the authors of the work (cf. Viana et al. 2009). The experiments were run on Intel/Celeron 1.6 GHz processor. The authors did not explain for the score and time columns whether the values denote best or average measures. They did not also mention how many processors were used in the experiment. Assuming that the columns present average values one notices, that our methods generate similar results for instances Angi, Ethe, Gris, better results for Golo, Ropa, Tenu, and worse results for Carp and Schu.

Table 11 The comparison of the proposed algorithms with related methods published in the literature

Instance	Best known	Method	Score			Time		Grasp/VND (Ribeiro and Vianna 2005)		PMN (Viana et al. 2009)		PPN (Goëffon et al. 2008)	
			Avg.	Std. dev.	Best	Avg.	Std. dev.	Best	Time	Score	Time	Score	Time
Angi	**216**	NNI+PV1	328.58	12.25	300	0.21	0.03	**216**	5099	217	138	218.0	1.7
		NNI+PV2	326.18	9.45	305	0.23	0.03						
		SPR+PV1	217.50	1.00	**216**	34.76	8.69						
		SPR+PV2	217.27	0.65	**216**	38.03	4.29						
		VNS1+PV1	217.50	1.17	**216**	19.51	8.07						
		VNS1+PV2	217.36	0.92	**216**	18.19	3.39						
Carp	**548**	NNI+PV1	1076.82	36.74	1028	9.92	1.37	**548**	82176	550	3719	554.8	3.1
		NNI+PV2	1059.82	37.38	996	9.26	1.63						
		SPR+PV1	551.30	1.70	**548**	2882.14	565.66						
		SPR+PV2	551.27	0.79	550	2753.96	497.86						
		VNS1+PV1	551.55	0.93	550	922.78	201.32						
		VNS1+PV2	551.82	1.47	550	858.33	83.04						
Ethe	**372**	NNI+PV1	529.00	21.20	495	0.92	0.14	**372**	10042	374	530	374	1.0
		NNI+PV2	535.18	19.88	505	0.90	0.17						
		SPR+PV1	373.64	0.92	**372**	115.07	17.43						
		SPR+PV2	374.09	0.83	**372**	113.28	18.46						
		VNS1+PV1	373.18	0.60	**372**	29.21	5.77						
		VNS1+PV2	373.55	0.93	**372**	29.17	5.37						
Golo	**496**	NNI+PV1	731.18	19.27	695	1.53	0.36	**496**	32836	502	1171	505.8	4.2
		NNI+PV2	733.18	13.92	718	1.58	0.38						
		SPR+PV1	501.36	2.01	499	397.51	83.56						
		SPR+PV2	500.18	2.14	497	333.20	69.60						
		VNS1+PV1	500.91	1.97	498	140.55	34.82						
		VNS1+PV2	501.73	1.35	500	159.75	28.46						
Gris	**172**	NNI+PV1	287.45	19.20	262	0.37	0.08	**172**	3505	**172**	226	**172.0**	1.7
		NNI+PV2	276.36	12.95	255	0.39	0.06						
		SPR+PV1	**172.00**	0.00	**172**	46.74	4.21						
		SPR+PV2	**172.00**	0.00	**172**	47.95	4.25						
		VNS1+PV1	**172.00**	0.00	**172**	14.61	2.43						
		VNS1+PV2	**172.00**	0.00	**172**	14.40	1.09						

Table 11 (*Continued*)

Instance	Best known	Method	Score			Time		Grasp/VND (Ribeiro and Vianna 2005)		PMN (Viana et al. 2009)		PPN (Goëffon et al. 2008)	
			Avg.	Std. dev.	Best	Avg.	Std. dev.	Best	Time	Score	Time	Score	Time
Ropa	**325**	NNI+PV1	553.10	14.07	527	1.21	0.24	**325**	15764	328	1038	328.7	1.6
		NNI+PV2	546.55	13.47	528	1.28	0.22						
		SPR+PV1	327.27	0.47	327	308.51	49.47						
		SPR+PV2	327.27	0.79	326	302.80	48.70						
		VNS1+PV1	327.91	1.04	326	107.18	22.61						
		VNS1+PV2	327.55	1.04	326	107.28	19.24						
Schu	**759**	NNI+PV1	1344.92	72.17	1236	14.42	1.91	**759**	113391	760	4917	768.6	3.5
		NNI+PV2	1339.55	84.26	1187	16.76	2.58						
		SPR+PV1	760.73	1.01	**759**	6047.63	1772.92						
		SPR+PV2	761.17	1.34	**759**	5319.59	471.99						
		VNS1+PV1	762.18	1.33	760	1311.84	276.04						
		VNS1+PV2	762.00	1.67	**759**	1152.96	243.99						
Tenu	**682**	NNI+PV1	786.27	41.31	727	2.81	0.35	**682**	7497	685	1299	686.6	2.6
		NNI+PV2	795.00	41.38	730	2.68	0.52						
		SPR+PV1	682.73	0.47	**682**	404.00	57.27						
		SPR+PV2	682.73	0.47	**682**	398.02	49.47						
		VNS1+PV1	682.91	0.94	**682**	44.92	8.04						
		VNS1+PV2	682.80	0.63	**682**	40.27	8.63						

Table 12 The Fitch scores of the trees found by different known methods for the first instance

	Method		
	DNAPARS	MEGA	LVB
Average	2650.30	2662.25	2650.70
Standard deviation	2.92	5.25	3.03

The differences between the quality are slight, but comparing the execution times of the algorithms in most cases our methods dominate the parallel multi-neighborhood local search heuristic algorithm.

The results obtained using Local Search with Progressive Tree Neighborhood published in Goëffon et al. (2008) are presented in column PPN. The tests were conducted on 2.4 GHz Opteron 850 processor. The average scores of our methods are better than these presented by the authors.

It is also interesting how the quality of the found solutions evolves in time, i.e. how it is related with the number of iterations of the algorithms. It has been checked for the variants of PV1 method and the results are shown in Fig. 4, where the relation of the Fitch score and the execution time is presented. The relation was observed for instance TI-1. The tests have been run on 16 processors and the computation times have been measured for the whole set of processors working in parallel. It can be seen that the Fitch score is rapidly improved at the beginning of computation, however attaining good, near optimum solution can take a lot of time. This experiment confirms the previous observation that the method based on VNS2 reach better solutions within a comparable limit of computation time.

To check the quality of our algorithms, in Table 12 the results obtained by different known sequential methods for the first instance were compared. The methods used in the comparison include: DNAPARS, MEGA and LVB. The Fitch scores of the trees found by the methods are shown. Each entry provides an average over 20 runs. DNAPARS (cf. Felsenstein 1989, 2005) estimates phylogenies by the parsimony method using DNA sequences. This program carries out unrooted parsimony (analogous to Wagner trees) on DNA sequences. The MEGA results were obtained using the Maximum Parsimony method (cf. Eck and Dayhoff 1966) and the Close-Neighbor-Interchange algorithm (cf. Nei and Kumar 2000) with search level 4 (cf. Tamura et al. 2007) in which the initial trees were obtained with the random addition of sequences (30 replications). LVB (cf. Barker 2004) uses parsimony to reconstruct phylogeny from a nucleotide alignment, using a simulated annealing heuristic search procedure.

Analyzing Table 4 and Table 12, we see that our proposed algorithms generate trees of good quality which in most cases are better than those ones generated by other known phylogenetic methods.

5 Conclusions

In this work, three adaptive memory programming parallel algorithms for phylogenetic tree construction have been presented. The algorithms have master-slave structure. The slave processes are local search procedures exploring a part of a solution space, namely a neighborhood of some starting solution determined by the master process. This starting solution is constructed on the basis of locally optimal trees already found, stored in a memory. Such an approach leads the slave processes to explore fragments of the solution space containing potentially good solutions. The algorithms described in the paper can be easily modified by changing the local search procedures executed by slave processors. Here, several versions of the methods differing in the way the solution space is searched have been tested and compared. The most important conclusion is that applying adaptive memory programming (AMP) gives better results than an algorithm, where the starting points are generated randomly. AMP in connection with parallel processing related to the sequential algorithm without AMP gives a superlinear speedup, proving high efficiency of the proposed approach. The quality of the solutions obtained was measured and tested against solutions from other methods. The tests again demonstrated the advantage of these parallel algorithms over the existing software packages.

Acknowledgements The research was partially supported by the grant of the Ministry of Science and Higher Education of Poland No. 519314635 and the Research Grant from Poznan Supercomputing and Networking Center. The first author acknowledges a support of the INRIA Rhone-Alpes grant.

References

Andreatta, A. A., & Ribeiro, C. C. (2005). Heuristics for the phylogeny problem. *Journal of Heuristics*, *86*, 429–447.

Barker, D. (2004). LVB: Parsimony and simulated annealing in the search for phylogenetic trees. *Bioinformatics*, *20*, 274–275.

Blazewicz, J., Formanowicz, P., Kedziora, P., & Wojciechowski, P. (2004). Parallel algorithms for evolutionary history reconstruction. In *Lecture notes in computer science: Vol.* 3019 (pp. 1138–1145). Berlin: Springer.

Day, W. H. E., Jonhson, D. S., & Sankoff, D. (1986). The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences*, *81*, 33–42.

Eck, R. V., & Dayhoff, M. O. (1966). Atlas of protein sequence and structure. In *National biomedical research foundation*. Maryland: Silver Springs.

Edwards, A. W. F., & Cavalli-Sforza, L. L. (1963). The reconstruction of evolution. *Annals of Human Genetics*, *27*, 105–106.

Edwards, A. W. F., & Cavalli-Sforza, L. L. (1964). Reconstruction of evolutionary trees. In V. H. Heywood & J. McNeill (Eds.), *Phonetic and phylogenetic classification, systematics association publish: Vol.* 6 (pp. 67–76) London.

Fayad, M., & Schmidt, D. (1997). Object-oriented application frameworks. *Communications of the ACM*, *40*, 32–38.

Felsenstein, J. (1989). PHYLIP—Phylogeny Inference Package (Version 3.2), *Cladistics*, 5.

Felsenstein, J. (2004). *Inferring phylogenies*. Sunderland: Sinauer Associates.

Felsenstein, J. (2005). PHYLIP (Phylogeny Inference Package) version 3.6. Distributed by the author, Department of Genome Sciences, University of Washington, Seattle.

Fitch, W. M. (1971). Toward defining the course of evolution: minimum change for a specified tree topology. *Systematic Zoology*, *20*, 406–416.

Foulds, L. R., & Graham, R. L. (1982). The Steiner problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, *3*, 43–49.

Goëffon, A., Richer, J. M., & Hao, J. K. (2005). Local search for the maximum parsimony problem. In *Lecture notes in computer science: Vol. 3612. ICNC '05—first international conference on natural computation* (pp. 678–683). Berlin: Springer.

Goëffon, A., Richer, J. M., & Hao, J. K. (2008). Progressive tree neighborhood applied to the maximum parsimony problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *5*(1).

Glover, F. (1989). Tabu search, Part I. *ORSA Journal on Computing*, *1*, 190–206.

Glover, F. (1997). Tabu search and adaptive memory programming—advances, applications and challenges. In Barr, Helgason, & Kenington (Eds.) *Advances in metaheuristics, optimization and stochastic modeling technologies* (pp. 1–75). Boston: Kluwer Academic.

Goloboff, P. (1999). Analyzing large data sets in reasonable times: solution for composite optima. *Cladistics*, *15*, 415–428.

Gusfield, D. (1997). *Algorithms on strings, trees, and sequences*. Cambridge: Cambridge University Press.

Hendy, M. D., & Penny, D. (1982). Branch and bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences*, *59*, 277–290.

Kedziora, P., Blazewicz, J., Formanowicz, P., Figlerowicz, M., Alejska, M., Jackowiak, P., Malinowska, N., & Fratczak, A. (2005). Computational methods in diagnostics of chronic hepatitis C. *Bulletin of the Polish Academy of Sciences. Technical Sciences*, *53*, 273–281.

Lin, Y., Fang, S., & Thorne, J. (2007). A tabu search algorithm for maximum parsimony phylogeny. *European Journal of Operational Research*, *176*, 1908–1917.

Luckow, M., & Pimentel, R. A. (1985). Numerical Wagner computer programs. *Cladistics*, *1*, 47–66.

Mladenović, N., & Hansen, P. (1997). Variable neighbourhood search. *Computers and Operations Research*, *24*, 1097–1100.

Nei, M., & Kumar, S. (2000). *Molecular evolution and phylogenetics*. New York: Oxford University Press.

Nelesen, S., Liu, K., Zhao, D., Linder, C. R., & Warnow, T. (2008). The effect of the guide tree on multiple sequence alignments and subsequent phylogenetic analyses. *Pacific Symposium on Biocomputing*, *13*, 25–36.

Nixon, K. C. (1999). The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics*, *15*, 407–414.

R Development Core Team (2008) A Language and Environment for Statistical Computing. http://www.R-project.org, R Foundation for Statistical Computing, ISBN 3-900051-07-0.

Roshan, U., Moret, B. M. E., Williams, T. L., & Warnow, T. (2004). Rec-I-DCM3: a fast algorithmic technique for reconstructing large phylogenetic trees. In *Proceedings of the IEEE computational systems bioinformatics conference*, 2004.

Ribeiro, C. C., & Vianna, D. S. (2005). A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. *International Transactions in Operational Research*, *12*, 325–338.

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

Setubal, J., & Meidanis, J. (1997). *Introduction to computational molecular biology*. Boston: PWS Publishing Company.

Stamatakis, A. (2004). *Distributed and parallel algorithms and systems for inference of huge phylogenetic trees based on the maximum likelihood method*. Ph.D. thesis, Technische Universität München, Germany.

Taillard, E. D., Gambardella, L. M., Gendreau, M., & Potvin, J. (2001). Adaptive memory programming: a unified view of metaheuristics. *European Journal of Operational Research*, *135*, 1–16.

Tamura, K., Dudley, J., Nei, M., & Kumar, S. (2007). MEGA4: molecular evolutionary genetics analysis (MEGA) software version 4.0. *Molecular Biology and Evolution*, *24*, 1596–1599.

Viana, G. V. R., Gomes, F. A. C., Meneses, C. N., & Ferreira, C. E. (2009). Parallelization of a multineighborhood local search heuristic for a phylogeny problem. *International Journal of Bioinformatics Research and Applications (IJBRA)*, *5*, 163–177.