Maximum Parsimony Phylogenetic Inference Using Simulated Annealing

Jean-Michel Richer, Eduardo Rodriguez-Tello and Karla E. Vazquez-Ortiz

Abstract The Maximum Parsimony (MP) problem aims at reconstructing a phylogenetic tree from DNA sequences while minimizing the total number of genetic transformations. In this paper we propose a carefully devised simulated annealing implementation, called SAMPARS (Simulated Annealing for Maximum PARSimony), for finding near-optimal solutions for the MP problem. Different possibilities for its key components and input parameter values were carefully analyzed and tunned in order to find the combination of them offering the best quality solutions to the problem at a reasonable computational effort. Its performance is investigated through extensive experimentation over well known benchmark instances showing that our SAMPARS algorithm is able to improve some previous best-known solutions.

Key words: Maximum Parsimony, Phylogenetic Trees, Simulated Annealing.

1 Introduction

One of the main problems in Comparative Biology consists in establishing ancestral relationships between a group of *n* species or homologous genes in populations of different species, designated as taxa. These ancestral relationships are usually represented by a binary rooted tree, which is called a phylogenetic tree or a phylogeny [19].

Jean-Michel Richer
LERIA, Université d'Angers, 2 Boulevard Lavoisier, 49045 Angers Cedex 01, France.
e-mail: `jean-michel.richer@univ-angers.fr`

Eduardo Rodriguez-Tello and Karla E. Vazquez-Ortiz
CINVESTAV-Tamaulipas, Information Technology Laboratory, Km. 5.5 Carretera Victoria-Soto La Marina, 87130 Victoria Tamps., Mexico.
e-mail: `{kvazquez,ertello}@tamps.cinvestav.mx`

In the past phylogenetic trees were inferred by using morphologic characteristics like color, size, number of legs, etc. Nowadays, they are reconstructed using the information from biologic macromolecules like DNA (deoxyribonucleic acid), RNA (ribonucleic acid) and proteins. The problem of reconstructing molecular phylogenetic trees has become an important field of study in Bioinformatics and has many practical applications in population genetics, whole genome analysis, and the search for genetic predictors of disease [20, 35].

Given a set $\mathscr{S} = \{S_1, S_2, \ldots, S_n\}$ composed by n sequences of length k over a predefined alphabet $\mathscr{A}$ (operational taxa previously aligned). A *binary rooted phylogenetic tree* $T = (V, E)$ is used to represent their ancestral relationships, it consists of a set of nodes $V = \{v_1, \ldots, v_r\}$ and a set of edges $E \subseteq V \times V = \{\{u, v\} | u, v \in V\}$. The set of nodes V ($|V| = (2n-1)$) is partitioned into two subsets: I, containing $n-1$ *internal nodes* (hypothetical ancestors) each one having 2 descendants; and L, composed of *n leaves*, i.e., nodes with no descendant.

The parsimony sequence $P_w = \{z_1, \cdots, z_k\}$ for each internal node $I_w \in I$, whose descendants are $S_u = \{x_1, \cdots, x_k\}$ and $S_v = \{y_1, \cdots, y_k\}$, is calculated with the following expression:

$$\forall i, 1 \leq i \leq k, z_i = \begin{cases} x_i \cup y_i, \text{ if } x_i \cap y_i = \emptyset \\ x_i \cap y_i, \text{ otherwise} \end{cases} . \tag{1}$$

Then, the parsimony cost of the sequence P_w is defined as follows:

$$\phi(P_w) = \sum_{i=1}^{k} C_i \qquad \text{where} \quad C_i = \begin{cases} 1, \text{ if } x_i \cap y_i = \emptyset \\ 0, \text{ otherwise} \end{cases} , \tag{2}$$

and the parsimony cost for the tree T is obtained as follows:

$$\phi(T) = \sum_{\forall w \in I} \phi(P_w) . \tag{3}$$

Thus, the Maximum Parsimony (MP) problem consists in finding a tree topology T^* for which $\phi(T^*)$ is minimum, i.e.,

$$\phi(T^*) = \min\{\phi(T) : T \in \mathscr{T}\} , \tag{4}$$

where $\mathscr{T}$ is the set composed by all the possible tree topologies (the search space of the problem).

There exist many different methods reported in the literature, to solve the problem of reconstructing phylogenetic trees. These can be classified in three main different approaches. *Distance methods* [13, 32], *Probabilistic methods* [12, 33] and *Cladistic methods* [7, 11]. In this paper we focus our attention in a cladistic method based on the Maximum Parsimony (MP) criterion, which is considered in the literature as one of the most suitable evaluation criterion for phylogenies [24, 34].

It has been demonstrated that the MP problem is NP-complete [17], since it is equivalent to the combinatorial optimization problem known as the Steiner tree problem on hypercubes, which is proven to be NP-complete [14].

The MP problem has been exactly solved for very small instances ($n \leq 10$) using a branch & bound algorithm (B&B) originally proposed by Hendy and Penny [18]. However, this algorithm becomes impractical when the number of studied species n increases, since the size of the search space suffers a combinatorial explosion. Therefore, there is a need for heuristic methods to address the MP problem in reasonable time.

Andreatta and Ribeiro [4] compared three greedy algorithms of different complexity: *1stRotuGbr*, *Gstep_wR* and *Grstep*. They concluded from their experiments that, *Gstep_wR* was more efficient than *1stRotuGbr*, but expending more computational time. *Grstep* achieved good results only when it was combined with a local search method. Even when these methods attained good quality solutions, they were still far away from the optimal solutions.

In 2003, Barker proposed a software, called LVB, which implemented a multi-start simulated annealing algorithm for solving the MP problem [5]. Later, an updated version of LVB was released in 2010 [6]. This new version adds a hill-climbing phase at the end of each simulated annealing search and a new stop condition.

Ribeiro and Viana [26] in 2005 applied a greedy randomized adaptive search procedure (GRASP) for solving the MP problem and showed that this algorithm had the best performance with respect to the state-of-the-art algorithms. Different evolutionary algorithms were also reported for the MP problem. Among them we found GA+PR+LS, a genetic algorithm hybridized with local search which employs path-relinking to implement a progressive crossover operator [27]. More recently Richer, Goëffon and Hao [28] introduced a memetic algorithm called Hydra which yields the best-known solutions for a set of 20 benchmark instances proposed in [25].

This paper aims at developing a new simulated annealing (*SA*) algorithm implementation (hereafter called SAMPARS) for finding near-optimal solutions for the MP problem under the Fitch's criterion. To achieve this, different possibilities for its key components were carefully designed and evaluated. The SAMPARS input parameter values yielding the best quality solutions to the problem at a reasonable computational effort were determined by employing a tunning methodology based on Combinatorial Interaction Testing. The performance of the new proposed implementation is investigated through extensive experimentation over 20 well known benchmark instances and compared with other existing state-of-the-art algorithms, showing that our algorithm is able to improve some previous best-known solutions.

The rest of this paper is organized as follows. In Section 2 the components of our SAMPARS implementation are discussed in detail. Then, three computational experiments are presented in Sect. 3 devoted to determine the best parameter settings for SAMPARS and to compare its performance with respect to LVB, an existing SA implementation [5, 6] and two other representative state-of-the-art algorithms: GA+PR+LS [27] and Hydra [15]. Finally, the last section summarizes the main contributions of this work and presents some possible directions for future research.

2 An Improved Implementation of a Simulated Annealing Algorithm

Simulated Annealing (SA) is a general-purpose stochastic optimization technique that has proved to be an effective tool for approximating globally optimal solutions to many NP-hard optimization problems. However, it is well known that developing an effective SA algorithm requires a careful implementation of some essential components and an appropriate tuning of the parameters used [21, 22].

In this section we present an improved implementation of a SA algorithm (Algorithm 1), that we called SAMPARS, for finding tree topologies (phylogenies) with near-optimal parsimony costs under the Fitch's criterion. The main difference of our implementation, with respect to LVB [5, 6], occurs in the neighborhood function (line 9) which has been tailored to fit the specificity of the MP problem. SAMPARS employs a composed neighborhood function combining standard neighborhood relations for trees with a stochastic descent algorithm on the current solution (see Sect. 2.4), while LVB randomly selects a neighbor $T' \in \mathscr{T}$ of the current solution T.

Algorithm 1: SAMPARS algorithm

```
   input: $\mathscr{N}$, $\phi$, $t_i$, $CL$, $\alpha$, $\beta$
1  $T \leftarrow$ GenerateInitialSolution()
2  $T^* \leftarrow T$
3  $j \leftarrow 0$
4  $t_j \leftarrow t_i$
5  repeat
6      $c \leftarrow 0$
7      while $c < CL$ do
8          $c \leftarrow c + 1$
9          $T' \leftarrow$ GenerateNeighbor($T, c, \mathscr{N}$)
10         $\Delta\phi \leftarrow \phi(T') - \phi(T)$
11         Generate a random $u \in [0,1]$
12         if $(\Delta\phi < 0)$ or $(e^{-\Delta\phi/t} > u)$ then
13             $T \leftarrow T'$
14             if $\phi(T') < \phi(T^*)$ then $T^* \leftarrow T'$
15         end
16     end
17     $j \leftarrow j + 1$
18     $t_j \leftarrow$ UpdateCurrentTemperature($t_{j-1}$)
19 until $< stop\ condition >$
20 return $T^*$
```

Next all the implementation details of the proposed SAMPARS algorithm are presented. For some of these components different possibilities were analyzed (see Sect. 3.2) in order to find the combination of them which offers the best quality solutions at a reasonable computational effort.

2.1 Internal Representation and Search Space

Let T be a potential solution in the search space $\mathscr{T}$, that is a phylogenetic tree representing the ancestral relationships of a group of n operational taxa, each one of length k over a predefined alphabet $\mathscr{A}$. Then T is represented as a binary rooted tree composed of $n-1$ internal nodes and n leaves. The size of the search space $|\mathscr{T}|$, i.e., the number of rooted tree topologies is given by the following expression [40]:

$$|\mathscr{T}| = (2n-3)!/2^{n-2}(n-2)! \tag{5}$$

2.2 Evaluation Function

The evaluation function is one of the key elements for the successful implementation of metaheuristic algorithms because it is in charge of guiding the search process toward good solutions in a combinatorial search space.

Previously reported metaheuristic methods for solving the MP problem have commonly evaluated the quality of a potential solution, $\phi(T)$, using the parsimony cost depicted in (3) [4, 5, 26–28]. In our SAMPARS implementation this evaluation function was also used.

2.3 Initial Solution

The initial solution is the starting phylogenetic tree used for the algorithm to begin the search of better configurations in the search space $\mathscr{T}$.

In the existing SA implementation for the MP problem [5, 6] the initial solution is randomly generated. In contrast, SAMPARS creates the starting solution using a greedy procedure that guarantees a better quality initial solution. The proposed procedure can be described as follows.

First, it generates a random permutation of the studied taxa, which is used to indicate the order in which the leaves (taxa) will be processed. Then, the root node of the tree is created and the first and second taxa in the permutation are binded to it. The rest of the taxa in the permutation are appended to the tree one by one. Each time a new taxon is added to the partial tree, the algorithm analyzes all the possible insertion positions in order to select the one that minimizes the increase in the tree's parsimony cost. This process is iterated until all the remaining taxa in the permutation are processed.

2.4 Neighborhood Functions

The most common practice in the reported metaheuristics for the MP problem [4, 26, 27] is employing one of the following three neighborhood functions. The first one, called Nearest Neighbor Interchange (NNI), was proposed by Waterman and Smith [39]. It exchanges two subtrees separated by an internal node. Given that each tree has $n-3$ internal nodes and two possibles moves by branch, then there exist $2n-6$ NNI neighboring solutions [29]. The second one, is known as Subtree Pruning and Regrafting (SPR) [36]. It cuts a branch of the tree and reinserts the resulting subtree elsewhere generating a new internal node. For each tree there exist $2(n-3)(2n-7)$ possible SPR neighboring solutions [3]. Finally, the Tree Bisection and Reconnection (TBR) [36] consists in dividing the tree into two subtrees that will be reconnected from one of their branches. From a given tree, the TBR neighborhood induces at most $(2n-3)(n-3)^2$ neighboring trees [3].

LVB, the existing SA implementation for the MP problem [5, 6] alternates the use of the NNI and SPR neighborhood functions at each iteration of the search process. In the case of our SAMPARS algorithm both the SPR and the TBR neighborhood relations are implemented. However, from our preliminary experiments it has been observed that the separated use of these neighborhood functions is not sufficient to reach the best-known solutions, because both of them are highly disruptive. In order to achieve a better performance for SAMPARS, we have decided to use a third complementary neighborhood structure. It is based on a stochastic descent algorithm with a best-improve scheme which is occasionally applied to the neighboring solution T' prior to returning it. Our compound neighborhood function is inspired by the ideas reported in [23], where the advantage of using this approach is well documented.

Algorithm 2: GenerateNeighbor

```
  input: $T$, $c$, $\mathcal{N}$
1 randomly select $T' \in \mathcal{N}(T)$        // A SPR or TBR neighboring solution
2 if c is a multiple of 15 then
3     $T' \leftarrow$ Descent($T'$)
4 end
5 return $T'$
```

2.5 Cooling Schedule

A cooling schedule is defined by the following parameters: an initial temperature, a final temperature or a stopping criterion, the maximum number of neighboring solutions that can be generated at each temperature (Markov chain length), and a rule for decrementing the temperature. The cooling schedule governs the convergence of the SA algorithm. At the beginning of the search, when the temperature is large, the probability of accepting solutions of worse quality than the current solution (uphill moves) is high. It allows the algorithm to escape from local minima. The probability to accept such moves is gradually decreased as the temperature goes to zero.

The literature offers a number of different cooling schedules, see for instance [1, 2, 30, 37]. They can be divided into two main categories: static and dynamic. In a static cooling schedule, the parameters are fixed and cannot be changed during the execution of the algorithm. With a dynamic cooling schedule the parameters are adaptively changed during the execution.

In our SAMPARS implementation we preferred a geometrical cooling scheme (static) mainly for its simplicity. It starts at an initial temperature t_i that can either be defined by the user or automatically computed using the following formula: $(k+n)^{(1.0/3.3)}$ which generates values under 6.0 for most of the tested benchmark instances. Then, this temperature is decremented at each round by a factor $\alpha = 0.99$ using the relation $t_j = \alpha t_{j-1}$. A reheat mechanism has also been implemented. If the best-so-far solution is not improved during $maxNITD = 50$ consecutive temperature decrements, the current temperature t_j is increased by a factor $\beta = 1.4$ using the function $t_j = \beta t_{j-1}$. In our implementation this reheat mechanism can be applied at most $maxReheat = 3$ times, since it represents a good trade-off between efficiency and quality of solution found.

For each temperature t_j, the maximum number of visited neighboring solutions is CL. It depends directly on the parameters n and k of the studied instance, since we have observed that more moves are required for bigger trees [38]. Next, we present the three different values that CL can take.

- small: $CL = 15(n+k)$
- medium: $CL = 23(n+k)$
- large: $CL = 40(n+k)$

The parameter values presented in this section were chosen based on the results obtained in a preliminary experimentation. For the reason of space limitation we did not present here these experiments.

2.6 Stop Condition

The SAMPARS algorithm stops if it ceases to make progress. In our implementation a lack of progress exists if after $\omega = 40$ (*frozen factor*) consecutive temperature decrements the best-so-far solution is not improved.

We will see later that thanks to the main features presented in this section, our SAMPARS algorithm reaches good quality results, which are sometimes better than the best-known solutions reported in the literature [15, 27].

3 Computational Experiments

In this section three main experiments were accomplished to evaluate the performance of the proposed SAMPARS algorithm and some of its components are presented. The objective of the first experiment is to determine both a component combination, and a set of parameter values which permit SAMPARS to attain the best trade-off between solution quality and computational effort. The purpose of the second experiment is to carry out a performance comparison of SAMPARS with respect to an existing SA algorithm called LVB [5, 6]. The third experiment is devoted to asses the performance of SAMPARS with respect to two representative state-of-the-art procedures: Hydra [15] and GA+PR+LS [27].

For these experiments SAMPARS was coded in C++ and compiled with g++ using the optimization flag -O3. It was run sequentially into a CPU Xeon X5650 at 2.66 GHz, 2 GB of RAM with Linux operating system. Due to the non-deterministic nature of the studied algorithms, 30 independent runs were executed for each of the selected benchmark instances in each experiment presented in this section.

3.1 Benchmark Instances and Performance Assessment

The test-suite that we have used in our experiments is the same proposed by Ribeiro and Vianna [26] and later employed in other works [15, 27]. It consists of 20 randomly generated instances with a number of sequences (n) ranging from 45 to 75 whose length (k) varies from 61 to 159.

For all the experiments, 30 independent executions were performed. The criteria used for evaluating the performance of the algorithms are the same as those used in the literature: the best parsimony cost found for each instance (smaller values are better) and the expended CPU time in seconds.

3.2 Components and Parameters Tunning

Optimizing parameter settings is an important task in the context of algorithm design. Different procedures have been proposed in the literature to find the most suitable combination of parameter values [10, 16]. In this paper we employ a tunning methodology based on Combinatorial Interaction Testing (CIT) [8]. We have decided to use CIT, because it allows to significantly reduce the number of tests (experiments) needed to determine the best parameter settings of an algorithm. Instead of exhaustive testing all the parameter value combinations of the algorithm, it only analyzes the interactions of t (or fewer) input parameters by creating interaction test-suites that include at least once all the t-way combinations between these parameters and their values.

Table 1 Input parameters of the SAMPARS algorithm and their selected values.

IS	α	t_i	$\mathscr{N}$	CL
Greedy	0.99	6.0	SPR	$15(n+k)$
Random	0.85	$(k+n)^{(1.0/3.3)}$	TBR	$23(n+k)$
-	-	-	-	$40(n+k)$

Covering arrays (CAs) are combinatorial designs which are extensively used to represent those interaction test-suites. A covering array, $\mathrm{CA}(N;t,k,v)$, of size N, strength t, degree k, and order v is an $N \times k$ array on v symbols such that every $N \times t$ sub-array includes, at least once, all the ordered subsets from v symbols of size t (t-tuples) [9]. The minimum N for which a $\mathrm{CA}(N;t,k,v)$ exists is the *covering array number* and it is defined according to the following expression: $\mathrm{CAN}(t,k,v) = min\{N : \exists \mathrm{CA}(N;t,k,v)\}$.

CAs are used to represent an interaction test-suite as follows. In an algorithm we have k input parameters. Each of these has v values or levels. An interaction test-suite is an $N \times k$ array where each row is a test case. Each column represents an input parameter and a value in the column is the particular configuration. This test-suite allows to cover all the t-way combinations of input parameter values at least once. Thus, the costs of tunning the algorithm can be substantially reduced by minimizing the number of test cases N in the covering array.

In practice, algorithms' input parameters do not have exactly the same number of values (levels). To overcome this limitation of CAs, mixed level covering arrays (MCAs) are used. A $\mathrm{MCA}(N;t,k,(v_1,v_2,\cdots,v_k))$ is an $N \times k$ array on v symbols ($v = \sum_{i=1}^{k} v_i$), where each column i ($1 \leq i \leq k$) of this array contains only elements from a set S_i, with $|S_i| = v_i$. This array has the property that the rows of each $N \times t$ sub-array cover all t-tuples of values from the t columns at least once. Next, we present the details of the tunning process, based on CIT, for the particular case of our SAMPARS algorithm.

First, we have identified $k = 5$ input parameters used for SAMPARS: initial solution procedure IS, cooling factor α, initial temperature t_i, neighborhood function $\mathscr{N}$ and maximum number of visited neighboring solutions CL. Based on some preliminary experiments, certain reasonable values were selected for each one of those input parameters (shown in Table 1).

The smallest possible mixed level covering array $\mathrm{MCA}(24;4,5,(2,2,2,2,3))$, shown (transposed) in Table 2, was obtained by using the Memetic Algorithm reported in [31]. This covering array can be easily mapped into an interaction test-suite by replacing each symbol from each column to its corresponding parameter value. For instance, we can map 0 in the first column (the first line in Table 2) to Greedy and 1 to Random. The resulting interaction test-suite contains, thus, 24 test cases (parameter settings) which include at least once all the 4-way combinations between SAMPARS's input parameters and their values[1].

Each one of those 24 test cases was used to executed 30 times the SAMPARS algorithm over the 20 instances of the test-suite described in Sect. 3.1. The data

[1] In contrast, with an exhaustive testing which contains $3(2^4) = 48$ test cases.

Table 2 Mixed level covering array $\mathrm{MCA}(24;4,5,(2,2,2,2,3))$ representing an interaction test-suite for tunning SAMPARS (transposed).

1	0	1	1	1	0	0	0	1	0	1	1	0	1	0	1	0	0	0	0	1	0	1	1
1	0	0	0	1	0	0	1	0	0	1	0	1	0	1	0	0	1	0	1	1	1	1	1
1	1	0	1	0	1	0	1	1	0	1	1	1	0	0	0	0	0	1	1	1	0	0	0
0	0	0	1	1	1	0	0	1	1	1	0	1	0	0	1	1	0	0	1	0	1	0	1
1	2	1	1	1	0	0	0	2	1	0	0	2	2	2	0	2	1	1	1	2	0	0	2

Table 3 Results from the 5 best parameter test cases in the tuning experiment.

Num.	Test case	Avg. parsimony	Avg. CPU time
17	**00012**	**1004.06**	**3512.51**
14	10002	1004.14	3511.35
10	00011	1005.00	2058.93
3	10001	1005.02	2047.52
9	10112	1005.29	3136.40

generated by these 14400 executions is summarized in Fig. 1, which depicts the average cost reached over the selected instances by each test case.

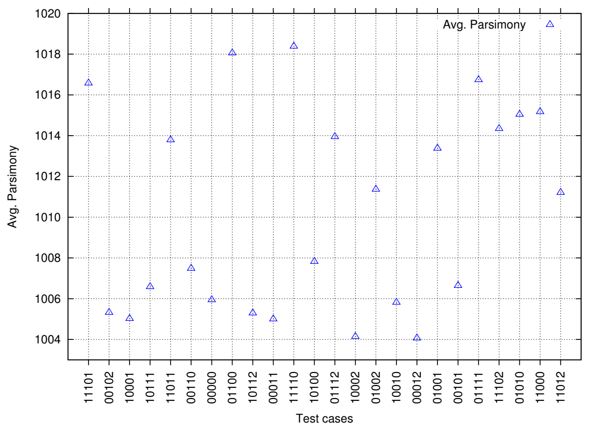

Fig. 1 Average results obtained in the tuning experiments using 24 test cases over 20 standard benchmark instances.

From this graphic we have selected the 5 test cases which yield the best results. Their average parsimony cost and the average CPU time in seconds are presented in Table 3. This table allowed us to observe that the parameter setting giving the best trade-off between solution quality and computational effort corresponds to the test case number 17 (shown in bold). The best average parsimony cost with an acceptable speed is thus reached with the following input parameter values: initial solution procedure $IS = Greedy$, cooling factor $\alpha = 0.99$, initial temperature $t_i = 6.0$, neighborhood function $\mathcal{N} = TBR$ and maximum number of visited neighboring solutions $CL = 40(n+k)$. These values are thus used in the experimentation reported next.

3.3 Comparing SAMPARS with an Existing SA Implementation

For this experiment a subset of six representative benchmark instances, taken from the test-suite described in Sect. 3.1, was selected (comparable results were obtained with all the other tested instances). Then, the last version of LVB was obtained, compiled and executed in our computational platform using the input parameters suggested by their authors [6]. SAMPARS were also executed over the same six instances.

Table 4 displays the detailed computational results produced by this experiment. The first three columns in the table indicate the name of the instance as well as its number of taxa (n) and length (k). For each compared algorithm the best (B), average (*Avg.*), and standard deviation (*Dev.*) of the parsimony cost attained in 30 independent executions and its average CPU time in seconds are listed in columns 4 to 11. A statistical significance analysis was performed for this experiment. First, *D'Agostino-Pearson's omnibus* K^2 test was used to evaluate the normality of data distributions. For normally distributed data, either *ANOVA* or the *Welch's t* parametric tests were used depending on whether the variances across the samples were homogeneous (*homoskedasticity*) or not. This was investigated using the *Bartlett's* test. For non-normal data, the nonparametric *Kruskal-Wallis* test was adopted. A significance level of 0.05 has been considered. The resulting *P-value* is presented in Column 12. Last column shows a "+" symbol if there exists a statistically significant increase in performance achieved by SAMPARS with regard to LVB, the existing SA Implementation.

Table 4 Comparison between SAMPARS and LVB (an existing SA implementation [5, 6]) over a subset of six selected instances.

			LVB				SAMPARS					
Instance	*n*	*k*	*B*	*Avg.*	*Dev.*	*T*	*B*	*Avg.*	*Dev.*	*T*	*P-value*	*SS*
tst01	45	61	549	553.87	2.47	85.57	545	545.83	0.87	295.80	1.80E-11	+
tst02	47	151	1367	1375.33	4.77	23.63	1354	1356.13	1.33	479.50	5.08E-21	+
tst03	49	111	840	850.83	4.86	68.02	833	834.00	1.05	577.12	1.38E-18	+
tst08	57	119	867	879.80	5.01	922.61	852	854.53	2.37	665.50	2.12E-11	+
tst09	59	93	1153	1160.77	4.60	58.53	1143	1145.50	1.11	719.24	3.48E-18	+
tst10	60	71	727	738.00	5.59	570.62	720	721.27	0.78	500.28	1.89E-16	+
Avg.			917.17	926.43	4.55	288.16	907.83	909.54	1.25	539.57		

From Table 4 we can observe that SAMPARS is the most time-consuming algorithm, since it uses an average of 539.57 seconds for solving these six instances. On the contrary, LVB employs only 288.16 seconds. However, we can also remark that SAMPARS can take advantage of its longer executions. Indeed it is able to consistently improve the best results found by LVB, obtaining in certain instances, like *tst08*, an important decrease in the parsimony cost (up to $-15 = 852 - 867$). Furthermore, the solution cost found by SAMPARS presents a relatively small standard deviation (see Column *Dev.*). It is an indicator of the algorithm's precision and robustness since it shows that in average the performance of SAMPARS does not present important fluctuations.

The statistical analysis presented in the last two columns of Table 4 confirms that there exists a statistically significant increase in performance achieved by SAMPARS with regard to LVB on the six studied instances. Thus, we can conclude that SAMPARS is more effective than the existing SA algorithm reported in [5, 6]. Below, we will present more computational results obtained from a performance comparison carried out between SAMPARS and some state-of-the-art procedures.

3.4 Comparing SAMPARS with the State-of-the-art Procedures

In this experiment a performance comparison of the best solutions achieved by SAMPARS with respect to those produced by GA+PR+LS [27] and Hydra [15] was carried out over the test-suite described in Sect. 3.1. The results from this experiment are depicted in Table 5. Columns 1 to 3 indicate the instance and its size in terms of taxa (n) and length (k). The best solutions found by GA+PR+LS and Hydra, in terms of parsimony cost Φ are listed in the next two columns. Columns 6 to 9 present the best (B), average ($Avg.$), and standard deviation ($Dev.$) of the parsimony cost attained by SAMPARS in 30 independent executions, as well as its average CPU time in seconds. Finally, the difference (δ) between the best result produced by our SAMPARS algorithm and the best-known solution produced by either GA+PR+LS or Hydra is shown in the last column.

Table 5 Performance comparison among SAMPARS, GA+PR+LS [27] and Hydra [15] over 20 standard benchmark instances.

					SAMPARS				
Instance	*n*	*k*	GA+PR+LS	Hydra	*B*	*Avg.*	*Dev.*	*T*	δ
tst01	45	61	547	545	545	545.13	0.43	1407.57	0
tst02	47	151	1361	1354	1354	1355.30	0.97	1938.23	0
tst03	49	111	837	833	833	833.43	0.56	2506.30	0
tst04	50	97	590	588	**587**	588.23	0.80	1341.17	-1
tst05	52	75	792	789	789	789.00	0.00	2007.90	0
tst06	54	65	603	596	596	596.57	0.56	1164.27	0
tst07	56	143	1274	1269	1269	1270.83	1.63	4063.80	0
tst08	57	119	862	852	852	853.33	1.27	2884.73	0
tst09	59	93	1150	1144	**1141**	1144.73	1.09	3237.53	-3
tst10	60	71	722	721	**720**	720.80	0.70	2288.00	-1
tst11	62	63	547	542	**541**	542.21	0.72	3807.79	-1
tst12	64	147	1225	1211	**1208**	1215.27	2.76	3668.40	-3
tst13	65	113	1524	1515	1515	1517.77	1.91	2514.20	0
tst14	67	99	1171	1160	1160	1163.03	1.82	2847.13	0
tst15	69	77	758	752	752	753.90	1.11	4808.63	0
tst16	70	69	537	529	529	531.00	1.23	3268.20	0
tst17	71	159	2469	2453	**2450**	2456.00	2.63	8020.23	-3
tst18	73	117	1531	1522	**1521**	1525.67	3.96	4451.37	-1
tst19	74	95	1024	1013	**1012**	1016.23	2.14	6875.30	-1
tst20	75	79	671	661	**659**	662.82	1.44	7149.43	-2
Avg.			1009.75	1002.45	1001.65	1004.06	1.39	3512.51	

The analysis of the data presented in Table 5 lead us to the following observations. First, we clearly remark that the procedure GA+PR+LS [27] consistently returns poorer quality solutions than Hydra and SAMPARS. Second, the best solution quality attained by the proposed SAMPARS algorithm is very competitive with respect to that produced by the existing state-of-the-art procedure called Hydra [15], since in average SAMPARS provides solutions whose parsimony costs are smaller (compare Columns 5 and 6). In fact, it is able to improve on 9 previous best-known solutions produced by Hydra and to equal these results for the other 11 benchmark instances.

Thus, as this experiment confirms, our SAMPARS algorithm is an effective alternative for solving the MP problem, compared with the two more representative state-of-the-art algorithms: GA+PR+LS [27] and Hydra [15].

4 Conclusions

In this paper we have presented an improved simulated annealing algorithm (SAMPARS) for finding near-optimal solutions for the MP problem under the Fitch's criterion. SAMPARS's components and parameter values were carefully determined, through the use of a tunning methodology based on Combinatorial Interaction Testing [8], to yield the best solution quality in a reasonable computational time.

Extensive experimentation was conducted to investigate the performance of SAMPARS over a set of 20 well known benchmark instances. In these experiments our algorithm was carefully compared with an existing simulated annealing implementation (LVB) [5, 6], and other two state-of-the-art algorithms. The results show that there exists a statistically significant increase in performance achieved by SAMPARS with respect to LVB. SAMPARS is in fact able to consistently improve the best results produced by LVB, obtaining in certain instances important reductions in the parsimony cost. Regarding GA+PR+LS [27], we have observed that in average this algorithm returns worse quality solutions than SAMPARS. Compared with the state-of-the-art algorithm called Hydra [15], our SAMPARS algorithm was able to improve on 9 previous best-known solutions and to equal these results on the other 11 selected benchmark instances. Furthermore, it was observed that the solution cost found by SAMPARS presents a relatively small standard deviation, which indicates the precision and robustness of the proposed approach. These experimental results confirm the practical advantages of using our algorithm for solving the MP problem.

Finding near-optimal solutions for the MP problem is a very challenging problem. However, the introduction of SAMPARS opens up an exciting range of possibilities for future research. One fruitful possibility is to analyze the use of different cooling schedules, stop conditions and mechanism for adapting the maximum number of visited neighboring solutions at each temperature depending on the behavior of the search process.

Acknowledgements This research work was partially funded by the following projects: 51623 Fondo Mixto CONACyT y Gobierno del Estado de Tamaulipas; CONACyT 99276, Algoritmos para la Canonización de Covering Arrays.

References

1. Aarts, E.H.L., Van Laarhoven, P.J.M.: Statistical cooling: A general approach to combinatorial optimization problems. Philips Journal of Research **40**, 193–226 (1985)
2. Abramson, D., Krishnamoorthy, M., Dang, H.: Simulated annealing cooling schedules for the school timetabling problem. Asia-Pacific Journal of Operational Research **16**(1), 1–22 (1999)
3. Allen, B.J., Steel, M.: Subtree transfer operations and their induced metrics on evolutionary trees. Annals of Combinatorics **5**(1), 1–15 (2001)
4. Andreatta, A., Ribeiro, C.C.: Heuristics for the phylogeny problem. Journal of Heuristics **8**(4), 429–447 (2002)
5. Barker, D.: LVB: parsimony and simulated annealing in the search for phylogenetic trees. Bioinformatics **20**(2), 274–275 (2003)
6. Barker, D.: LVB homepage (2012). URL `http://biology.st-andrews.ac.uk/cegg/lvb.aspx`
7. Cavalli-Sforza, L.L., Edwards, A.W.F.: Phylogenetic analysis. models and estimation procedures. The American Journal of Human Genetics **19**(3 Pt 1), 233–257 (1967)
8. Cohen, D.M., Dalal, S.R., Parelius, J., Patton, G.C.: The combinatorial design approach to automatic test generation. IEEE Software **13**(5), 83–88 (1996)
9. Colbourn, C.J.: Combinatorial aspects of covering arrays. Le Matematiche **58**, 121–167 (2004)
10. de Landgraaf, W.A., Eiben, A.E., Nannen, V.: Parameter calibration using meta-algorithms. In: In proceedings of the IEEE Congress on Evolutionary Computation, pp. 71–78. IEEE Press (2007)
11. Edwards, A.W.F., Cavalli-Sforza, L.L.: The reconstruction of evolution. Heredity **18**, 553 (1963)
12. Felsenstein, J.: Evolutionary trees from DNA sequences: a maximum likelihood approach. Journal of Molecular Evolution **17**(6), 368–376 (1981)
13. Fitch, W.M., Margoliash, E.: A method for estimating the number of invariant amino acid coding positions in a gene using cytochrome c as a model case. Biochemical Genetics **1**(1), 65–71 (1967)
14. Garey, M.R., Johnson, D.S.: The rectilinear Steiner tree problem is NP-Complete. SIAM Journal on Applied Mathematics **32**(4), 826–834 (1977)
15. Goëffon, A.: Nouvelles heuristiques de voisinage et mémétiques pour le problème maximum de parcimonie. Ph.D. thesis, LERIA, Université d'Angers (2006)
16. Gunawan, A., Lau, H.C., Lindawati: Fine-tuning algorithm parameters using the design of experiments. Lecture Notes in Computer Science **6683**, 131–145 (2011)
17. Gusfield, D.: Algorithms on strings, trees, and sequences: Computer science and computational biology, 1st. edn. Cambridge University Press (1997)
18. Hendy, M.D., Penny, D.: Branch and bound algorithms to determine minimal evolutionary trees. Mathematical Biosciences **59**(2), 277–290 (1982)
19. Hennig, W.: Phylogenetic systematics. Phylogeny. University of Illinois Press, Urbana (1966)
20. Hillis, D.M., Moritz, C., Mable, B.K.: Molecular systematics, 2nd. edn. Sinauer Associates Inc., Sunderland, MA (1996)
21. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. Operations Research **37**(6), 865–892 (1989)

22. Johnson, D.S., Aragon, C.R., McGeoch, L.A., Schevon, C.: Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. Operations Research **39**(3), 378–406 (1991)
23. Lü, Z., Hao, J.K., Glover, F.: Neighborhood analysis: A case study on curriculum-based course timetabling. Journal of Heuristics **17**(2) (2011)
24. Penny, D., Foulds, L.R., Hendy, M.D.: Testing the theory of evolution by comparing phylogenetic trees constructed from five different protein sequences. Nature **297**, 197–200 (1982)
25. Ribeiro, C.C., Vianna, D.S.: A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. In: In proceedings of the II Workshop Brasileiro de Bioinformática, pp. 97–102. Macaé, Brazil (2003)
26. Ribeiro, C.C., Vianna, D.S.: A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. International Transactions in Operational Research **12**(3), 325–338 (2005)
27. Ribeiro, C.C., Vianna, D.S.: A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. International Transactions in Operational Research **16**(5), 641–657 (2009)
28. Richer, J.M., Goëffon, A., Hao, J.K.: A memetic algorithm for phylogenetic reconstruction with maximum parsimony. Lecture Notes in Computer Science **5483**, 164–175 (2009)
29. Robinson, D.F.: Comparison of labeled trees with valency three. Journal of Combinatorial Theory, Series B **11**(2), 105–119 (1971)
30. Rodriguez-Tello, E., Hao, J.K., Torres-Jimenez, J.: An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. Computers & Operations Research **35**(10), 3331–3346 (2008)
31. Rodriguez-Tello, E., Torres-Jimenez, J.: Memetic algorithms for constructing binary covering arrays of strength three. Lecture Notes in Computer Science **5975**, 86–97 (2010)
32. Saitou, N., Nei, M.: The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular Biology and Evolution **4**(4), 406–425 (1987)
33. Skourikhine, A.: Phylogenetic tree reconstruction using self-adaptive genetic algorithm. In: Proceedings of the IEEE International Symposium on Bio-Informatics and Biomedical Engineering, pp. 129–134. Arlington, VA , USA (2000)
34. Sober, E.: The nature of selection: Evolutionary theory in philosophical focus. University Of Chicago Press (1993)
35. Sridhar, S., Lam, F., Blelloch, G.E., Ravi, R., Schwartz, R.: Direct maximum parsimony phylogeny reconstruction from genotype data. BMC Bioinformatics **8**(472) (2007)
36. Swofford, D.L., Olsen, G.J., Waddell, P.J., Hillis, D.M.: Phylogeny reconstruction. In: Molecular Systematics, 2nd. edn., chap. 11, pp. 411–501. Sinauer Associates, Inc., Sunderland, MA (1996)
37. Van Laarhoven, P.J.M., Aarts, E.H.L.: Simulated annealing: Theory and applications. Kluwer Academic Publishers (1988)
38. Vazquez-Ortiz, K.E.: Metaheurísticas para la resolución del problema de máxima parsimonia. Master's thesis, LTI, Cinvestav - Tamaulipas, Cd. Vitoria, Tamps. Mexico (2011)
39. Waterman, M.S., Smith, T.F.: On the similarity of dendrograms. Journal of Theoretical Biology **73**(4), 784–800 (1978)
40. Xiong, J.: Essential Bioinformatics, 1st. edn. Cambridge University Press (2006)