METAHEURISTICS FOR THE MAXIMUM PARSIMONY PROBLEM

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

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 two different metaheuristic algorithm for finding near-optimal solutions for the MP problem are proposed: iterated local search (ILS) and simulated annealing (SA). Different possibilities for the key components of these algorithms were carefully analyzed in order to find the combination of them offering the best quality solutions to the problem at a reasonable computational effort. The performance of both metaheuristics is investigated through extensive experimentation over well known benchmark instances showing that our SA algorithm is able to improve some previous best-known solutions.

KEY WORDS
Maximum Parsimony, Phylogenetic Trees, Metaheuristics.

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 [1].

In the past the 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 biology and medicine [2].

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* $T = (V, E)$ for representing their ancestral relationships 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 $|V| = (2n - 1)$ 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 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:

$$\forall i, 1 \leq i \leq k, z_i = \begin{cases} x_i \cup y_i, & \text{if } x_i \cap y_i = \emptyset \\ x_i \cap y_i, & \text{otherwise} \end{cases} \quad (1)$$

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

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

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

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

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

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

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

There exist many different methods, reported in the literature, to solve the problem of reconstructing phylogenetic trees. These can be classified in three different approaches. *Distance methods* [3, 4], *Probabilistic methods* [5] and *Cladistic methods* [6, 7]. 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 the most suitable evaluation criterion for phylogenies [8, 9].

It has been demonstrated that the MP problem is NP-complete [10], since it is equivalent to the combinatorial optimization problem known as the Steiner tree problem on hypercubes, which is proven to be NP-complete [11]. Furthermore, the MP problem is highly combinatorial since the size of its search space $|\mathscr{T}|$, i.e., the number of tree topologies is given by the following expression [12]:

$$(2n - 3)!/2^{n-2}(n - 2)! \quad (5)$$

where n is the number of studied species.

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 y Penny [13]. 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 [14] 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.

Later, Ribeiro and Viana [15] 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 memetic algorithms were also reported for the MP problem. Among them we found the work of Matsuda [16] and Lewis [17]. More recently Richer, Goëffon and Hao [18] introduced a memetic algorithm called Hydra which yields the best-known solutions.

This paper aims at developing a metaheuristic algorithm for finding near-optimal solutions for the MP problem under the Fitch's criterion. To achieve this, we have designed and evaluated two different metaheuristics: iterated local search (*ILS*) and simulated annealing (*SA*). Different possibilities for the main components of these algorithms were carefully analyzed in order to find the combination which offers the best quality solutions to the problem at a reasonable computational effort. The performance of both metaheuristics is assessed with a test-suite, composed by 18 benchmark instances taken from the literature. The computational results are reported and compared with previously published ones, showing that our SA 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 iterated local search (*ILS*) algorithm are discussed in detail. Then, our simulated annealing (*SA*) algorithm and its implementation details are presented in Section 3. Section 4 is devoted to computational experiments carried out to identify the best component combination for both metaheuristics. Finally, the last section summarizes the main contributions of this work and presents some possible directions for future research.

2 Iterated Local Search

Iterated local search (*ILS*) is a metaheuristic algorithm which has been first proposed by Martin et al. [19] and generalized later in [20]. It starts by applying a local search algorithm to an initial solution. Then, at each iteration, the local optima obtained is perturbed. Local search is applied to this perturbed solution to generate a new solution which can be accepted as the new current solution under certain conditions. This iterative procedure repeats until a given stop condition is met. In the case of our ILS algorithm, the search process stops when the maximum number of non-improving neighboring solutions $maxCS$ allowed is reached (see Algorithm 1).

Algorithm 1: Iterated local search (*ILS*)

```
input: Neighborhood function 𝒩, evaluation function f,
       maximum non-improving neighboring solutions
       maxCS
s ← GenerateInitialSolution();
s'' ← LocalSearch(s);
if f(s'') < f(s) then s ← s'';
CS ← 0 ;
while CS < maxCS do
    s' ← Perturbation (s);
    s'' ← LocalSearch(s');
    if f(s'') < f(s) then
        s ← s'';
        CS ← 0;
    end
    else CS ← CS + 1
end
return s
```

Our ILS algorithm has five essentials components: initial solution, local search method (embedded heuristic), perturbation, acceptation criterion and stop condition. Next, we present the implementation details of these components.

2.1 Initial Solution

The implementation of a metaheuristic to solve the MP problem requires an initial solution. In this work we describe two different methods to generate initial solutions: random and greedy.

2.1.1 Random Method

This method constructs a phylogenetic tree by assigning each taxon (leaf) in a randomly selected position of the tree. This method starts by generating a random permutation of the taxa. This permutation indicates the order in which each leaf will be added to the tree. Then, the elements of this permutation are put into the tree in randomly selected positions. Next we show an example to clarify this procedure.

Suppose that we have an instance with four taxa (leaves) $\{H_0, H_1, H_2, H_3\}$, the first step consists in generating a random permutation of them, consider for instance the following permutation: $\{H_2, H_3, H_0, H_1\}$. Then, the method creates the root node of the tree and the first and second taxa of the permutation are binded to it (see Fig. 1).

Each time a new leaf (taxon) is added to the tree, an internal node is also generated in order to conserve the binary tree restriction. These new leaves are placed on the tree in a randomly selected position. Beginning from the

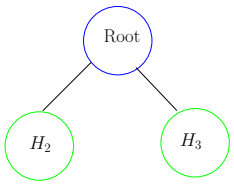

Figure 1. Creating the root of the tree.

root node, the algorithm randomly chooses either the right or the left branch of the tree until a leaf is found. In this position an internal node is created and the new taxon is inserted (see Fig. 2).

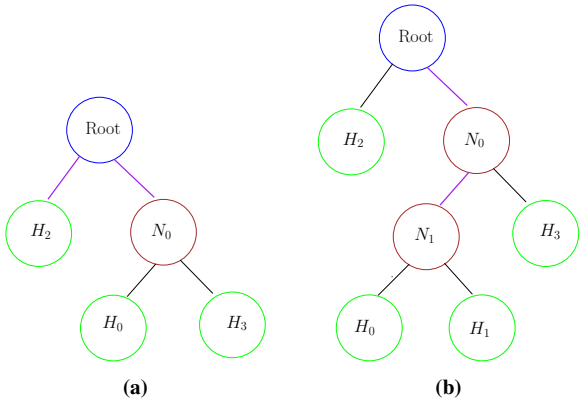

Figure 2. (a) Possible insertion sites for the taxon H_1, (b) Internal node N_1 was generated for inserting taxon H_1.

2.1.2 Greedy Method

The proposed greedy method for generating initial solutions proceeds as follows. First, it generates a random permutation of the studied taxa that indicates the order in which the leaves (taxa) will be processed. The change in the cost of the tree, produced by the insertion of the current taxon in all its possible positions is analyzed. Then, the current taxon is inserted in the position which minimizes the increase in the tree's parsimony.

For instance, consider the following permutation for an instance having four taxa: $\{H_2, H_3, H_0, H_1\}$. Our greedy algorithm creates the root node of the tree and the first and second taxa of the permutation are binded to it, just as in the random method (see Fig. 1).

Then, each time a new taxon (leaf) is added to the tree, the algorithm analyzes all the possible positions where it can be located. In Fig. 3(a) the leaves tagged with letter H are considered the possibles insertion sites. Suppose the cost of the tree in this figure is currently 20, and that it increases according to the position in which the new leaf H_1 is inserted as follows: H_2 27, H_0 25 and H_3 24. Given that we are trying to minimize the parsimony cost, the best position for the new taxon H_1 is in H_3 because it results in a parsimony cost of 23 (see Fig. 3(b)).

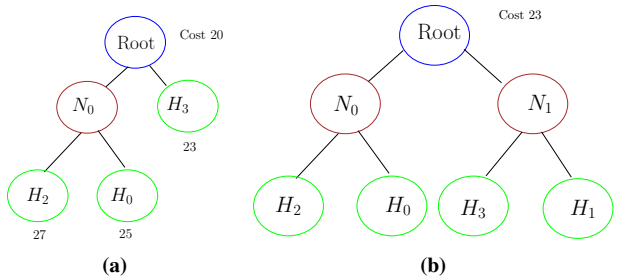

Figure 3. (a) Possible insertion sites for the taxon H_1 and its costs, (b) Internal node N_1 was generated for inserting taxon H_1 resulting in a cost of 23.

2.2 Local Search

The local search phase of our ILS algorithm is carried out using a Descent Algorithm (*DA*). We have selected this local search algorithm mainly because of its implementation simplicity and by the fact that it employs a single parameter, the stop condition. The search process in our DA stops when the maximum number of non-improving neighboring solutions allowed $maxCS$ is reached. For our experiments presented in Section 4 $maxCS$ was fixed to 100. Algorithm 2 presents the pseudo-code of DA.

The neighborhood function is one of the critical elements for the performance of any local search algorithm. In DA we have implemented four different neighborhood functions, two previously reported and two new combinations of them. Next, we briefly describe these neighborhood relations.

$\mathcal{N}_1$ implements the Nearest Neighbor Interchange (NNI) function proposed by Waterman and Smith [21]. It exchanges two subtrees separated by an internal node. Each tree have $2n - 6$ NNI neighboring solutions, $n - 3$ internal nodes and two possibles moves by edge [22].

$\mathcal{N}_2$ represents the Subtree Pruning and Regrafting (SPR) function originally reported by Swofford and Olsen [23]. It deletes an internal node with its descendants and reinserts this subtree in another position generating a new internal node. For each tree there exists $2(n-3)(2n-7)$ possible SPR neighboring solutions [24].

$\mathcal{N}_3$ and $\mathcal{N}_4$ are two different combinations of the neighboring functions described above. These combinations dynamically interchange between functions $\mathcal{N}_1$ and $\mathcal{N}_2$ on the basis of the current number of non-improving neighboring solutions CS as described in Table 1.

2.3 Neighborhood Functions

In our implementation of the ILS metaheuristic, we used seven different compound neighborhood functions, which

Neighborhood	$CS < 50$	$CS \geq 50$
$\mathcal{N}_1$	NNI	NNI
$\mathcal{N}_2$	SPR	SPR
$\mathcal{N}_3$	NNI	SPR
$\mathcal{N}_4$	SPR	NNI

Table 1. Neighborhood functions used by DA

Algorithm 2: Descent algorithm (DA)

input: Neighborhood function $\mathcal{N}$, evaluation function f, initial solution s, maximum non-improving neighboring solutions $maxCS$

$CS \leftarrow 0$;
while $CS < maxCS$ **do**
  Select $s' \in \mathcal{N}(s)$;
  if $f(s') < f(s)$ **then**
    $s \leftarrow s'$;
    $CS \leftarrow 0$;
  end
  else $CS \leftarrow CS + 1$
end
return s

combines the two basic neighborhood relations $\mathcal{N}_1$ and $\mathcal{N}_2$ in the following way.

The first four combinations are probabilistic, they apply the neighborhood $\mathcal{N}_1$ with probability p and the neighborhood $\mathcal{N}_2$ at a $(1.0 - p)$ rate. Table 2 shows these four compound neighborhood functions and its associated probability values.

Neighborhood	Probability p
$\mathcal{N}_5$	0.50
$\mathcal{N}_6$	0.30
$\mathcal{N}_7$	0.40
$\mathcal{N}_8$	0.70

Table 2. Probabilistic neighborhood functions used by our ILS implementation.

The other three combinations of the neighborhood functions $\mathcal{N}_1$ and $\mathcal{N}_2$ start using one of these functions and switch to the other depending on the value of the current number of non-improving neighboring solutions CS as described in Table 3.

2.4 Perturbation

The perturbation operator in a ILS method can be seen as a large random move of the current solution. The perturbation method should conserve some part of the solution and perturb strongly another part of it to move hopefully to another region of the search space.

The perturbation phase in our ILS algorithm starts by randomly selecting two different leaves of the tree. Then, the positions of these leaves are interchanged to produce a different tree (see Fig. 4).

Neighborhood	$\mathcal{N}_1$	$\mathcal{N}_2$
$\mathcal{N}_9$	$CS \geq 20$	$CS < 20$
$\mathcal{N}_{10}$	$CS < 20$	$CS \geq 20$
$\mathcal{N}_{11}$	$40 < CS < 60$	$CS \leq 40$ or $CS \geq 60$

Table 3. Combined neighborhood functions used by ILS, which employ the number of non-improving neighboring solutions CS.

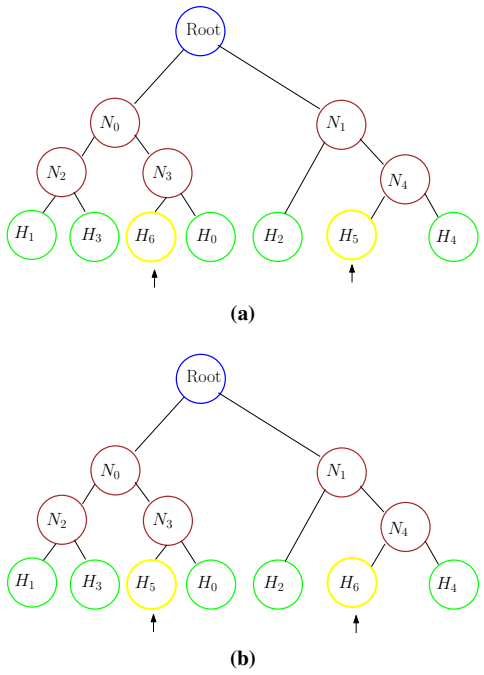

Figure 4. Example of a perturbation. (a) Selection of two different leaves of the tree, (b) Interchange of the two selected leaves of the tree.

2.5 Acceptance Criterion

The acceptance criterion defines the conditions that the new solution produced by the local search method must satisfy to replace the current solution.

In our ILS algorithm the new solutions s'' produced by the local search method are only accepted if they improve the cost of the current solution s.

2.6 Stop Condition

Our ILS algorithm stops when it ceases to make progress. In the proposed implementation a lack of progress exists when the current number of non-improving neighboring solutions CS reaches the limit $maxCS = 100$.

3 Simulated Annealing

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 [25, 26]. The main idea behind this metaheuristic consists in occasionally replacing the current solution with a non-improving neighboring solution in a controlled way. The probability of accepting this kind of non-improving moves is reduced as the search process is executed. Algorithm 3 presents the pseudo-code of this metaheuristic.

SA starts by generating an initial solution $s \in \mathscr{T}$ and setting the current temperature parameter t_k to an initial value t_i. Each iteration of this algorithm consists in randomly choosing a neighboring solution $s' \in \mathscr{N}(s)$ of the current solution s. Then, it is evaluated in order to decide if s' should replace s using the Metropolis criterion. This stochastic criterion depends on the current value of the temperature parameter t_k and the difference of cost Δf between the neighboring solution s' and the current solution s. A move from s to the solution s' is accepted if one of the following conditions is met: a) it improves the cost of the current solution, i.e., $\Delta f < 0$, or b) it worsens the cost of the current solution ($\Delta f \geq 0$) but the probability of accepting this move is greater than a random value u between 0 and 1 ($e^{-\Delta f/T_k} > u$).

Algorithm 3: Simulated annealing algorithm (SA)

```
input: N, f, t_i, t_f, l, α
s ← GenerateInitialSolution();
s* ← s ;
t_k ← t_i ;
while t_k > t_f do
    c ← 0;
    while c < l do
        c ← c + 1;
        Select s' ∈ N(s);
        Δf ← f(s') − f(s);
        Generate a random u ∈ [0, 1];
        if (Δf < 0) or (e^{−Δf/t_k} > u) then
            s ← s';
            if f(s') < f(s*) then s* ← s';
        end
    end
    t_k ← α t_{k−1};
end
return s* ;
```

3.1 Initial Solution

The initial solution for our SA algorithm is obtained using the random and greedy methods presented in Section 2.1.

3.2 Neighborhood Functions

We have implemented eleven compound neighborhood functions for our SA algorithm by combining the two basic neighborhood relations $\mathscr{N}_1$ and $\mathscr{N}_2$ described in Section 2.2. The first three combined neighborhood functions switch between the neighborhood functions $\mathscr{N}_1$ and $\mathscr{N}_2$ depending on the value of the current temperature t_k as described in Table 4, where $t_m = (t_i - t_f)/2$ and $t_a = (t_i - t_f)/4$.

Neighborhood	$\mathscr{N}_1$	$\mathscr{N}_2$
$\mathscr{N}_{12}$	$t_k \leq t_m$	$t_k > t_m$
$\mathscr{N}_{13}$	$t_k > t_m$	$t_k \leq t_m$
$\mathscr{N}_{14}$	$t_m \leq t_k > t_a$	$t_k > t_m$ or $t_a > t_k \geq t_f$

Table 4. Combined neighborhood functions used by SA employing the value of the current temperature t_k as a combination criterion.

Three other neighborhood functions were implemented which combine $\mathscr{N}_1$ and $\mathscr{N}_2$ depending on the number of visited neighboring solutions c at a given temperature t_k. The criteria followed to combine these neighborhood functions is detailed in Table 5, where l is the maximum number of visited solution at temperature t_k, $HM = l/2$ and $TM = 3l/4$.

Neighborhood	$\mathscr{N}_1$	$\mathscr{N}_2$
$\mathscr{N}_{15}$	$c \geq HM$	$c < HM$
$\mathscr{N}_{16}$	$c < HM$	$c \geq HM$
$\mathscr{N}_{17}$	$TM > c \geq HM$	$c < HM$ or $l \geq c \geq TM$

Table 5. Combined neighborhood functions used by SA employing the number of visited neighboring solutions c at a given temperature t_k as a combination criterion.

The last five neighborhood combinations are probabilistic, they apply the neighborhood $\mathscr{N}_1$ with probability p and the neighborhood $\mathscr{N}_2$ at a $(1.0 - p)$ rate. Table 6 shows these five compound neighborhood functions and its associated probability values.

Neighborhood	Probability p
$\mathscr{N}_{18}$	0.50
$\mathscr{N}_{19}$	0.45
$\mathscr{N}_{20}$	0.40
$\mathscr{N}_{21}$	0.35
$\mathscr{N}_{22}$	0.30

Table 6. Probabilistic neighborhood functions used by our SA implementation.

3.3 Cooling Schedule

The annealing schedule determines the degree of uphill movement permitted during the search and is, thus, critical to the algorithm's performance. The literature offers a number of different cooling schedules [27]. In our SA implementation we preferred a geometrical cooling scheme mainly for its simplicity. It starts at an initial temperature $t_i = \sqrt[2.5]{n+k}$. It allows the algorithm to start accepting approximatively 60% of the neighboring solutions. Then, at each round, the algorithm decrements the current temperature by a factor of $\alpha = 0.99$ using the relation $t_k = \alpha t_{k-1}$. For each temperature, the maximum number of visited neighboring solutions is $l = 10000$.

3.4 Stop Condition

Our SA algorithm stops if the current temperature reaches the value $t_f = 0.1$.

4 Experimentation and Results

In this section we present a series of experiments that we have carried out to determine which of the analyzed metaheuristics (ILS and SA) for finding near-optimal solutions for the MP problem has the better performance.

4.1 Benchmark Instances

In this research work we have used 18 benchmark instances organized in two sets. The first group consists of 8 real instances commonly used in the literature [14, 15, 28–31]. The characteristics for these instances are shown in the Table 7.

Instance	Taxa (n)	Length (k)	Best-known
ANGI	49	59	216
CARP	117	110	548
ETHE	58	86	372
GOLO	77	97	496
GRIS	47	93	172
ROPA	75	82	325
SCHU	113	146	759
TENU	56	179	682

Table 7. Eight real problem instances commonly used in the literature.

The second group consists of 10 instances which were randomly generated by Ribeiro and Vianna [15, 32] and later used by Richer et al. [18]. Table 8 shows the characteristics of this second group of benchmark instances.

Instance	Taxa (n)	Length (k)	Best-known
tst01	45	61	545
tst02	47	151	1356
tst03	49	111	833
tst04	50	97	588
tst05	52	75	789
tst06	54	65	596
tst07	56	143	1270
tst08	57	119	853
tst09	59	93	1145
tst10	60	71	721

Table 8. Ten randomly generated problem instances reported in [32].

In both tables the best-known solutions were taken from [18], which reports a memetic algorithm, called Hydra, considered as the state-of-the-art algorithm.

4.2 Experimental Conditions and Comparison Criteria

All our experiments were performed under similar conditions. We executed all the algorithms, 100 times with each problem instance. All the algorithms were coded in C and were compiled with gcc using the optimization flag -O3. They were run sequentially into a cluster of 4 processors Xeon six core X5650 at 2.66GHZ, 32 Gb of RAM and Linux operating system.

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.

4.3 Initialization Methods Comparison

The main objective of this first experiment is to determine what is the best method for generating initial solutions for their use in the analyzed metaheuristics. In order to accomplish it, we have tested the two different initialization methods presented in Section 2.1. Table 9 shows the best parsimony costs obtained for each method over each instance as well as the percent deviation to the best-known result reported in the literature (Δ).

Instance	Random	Δ	Greedy	Δ	Best-known
ANGI	394	0.82	229	0.06	216
CARP	1443	1.63	604	0.10	548
ETHE	815	1.19	393	0.06	372
GOLO	844	0.70	538	0.08	496
GRIS	414	1.41	184	0.07	172
ROPA	677	1.08	351	0.08	325
SCHU	2360	2.11	829	0.09	759
TENU	1442	1.11	726	0.06	682
tst01	684	0.25	582	0.07	545
tst02	1588	0.17	1419	0.05	1356
tst03	1036	0.24	891	0.07	833
tst04	762	0.29	639	0.09	588
tst05	988	0.25	833	0.06	789
tst06	787	0.32	646	0.08	596
tst07	1548	0.21	1356	0.07	1270
tst08	1080	0.27	921	0.08	853
tst09	1412	0.23	1209	0.06	1145
tst10	962	0.33	781	0.08	721
Average		0.700		0.073	

Table 9. Performance comparison between two different initialization methods.

From this table we can observe that the results provided by the greedy method were consistently better than those produced by the random method. Indeed, the results of the greedy method are in average 58.51% better than those achieved by the random initialization method. Furthermore, these results are produced expending a computational time equivalent to that consumed by the random initialization method. The excellent behavior of the greedy method is better observed in Fig. 5, which presents the results obtained by each compared method with respect to the best-known solutions.

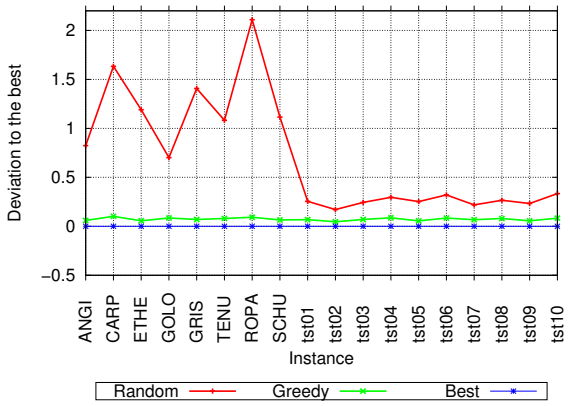

Figure 5. Performance comparison between two different initialization methods with respect to the best-known solutions.

4.4 Metaheuristic Algorithms Comparison

Two different metaheuristic algorithms were proposed in this work for solving the maximum parsimony problem: ILS and SA. For each one of these algorithms we have analyzed different possibilities for their key components. In particular, the influence of 2 initialization procedures and 22 neighborhood functions over the performance of these algorithms was extensively tested. These experiments leaded us to discover the components combination for each algorithm which yields the best possible performance. Given the space restrictions we did not present these experiments in detail. Table 10 summarizes the best components combination for each studied metaheuristic.

Algorithm	Initialization	Neighborhood
ILS	greedy	$\mathcal{N}_{11}$
SA	greedy	$\mathcal{N}_{20}$

Table 10. Components that provide the best performance for the proposed metaheuristic algorithms.

The rest of this section aims at presenting the experimental results obtained with the analyzed algorithms using their best components combinations. Table 11 shows the best results obtained by these metaheuristics and its percent deviation to the best-known solutions (Δ). A $\Delta = 0.0$ means that the best-known solution was reached, while a $\Delta < 0.0$ indicates that the best-known result was improved. We have also included the DA algorithm in these experiments for comparison purposes.

From Table 11 we can observe that the best performance is obtained by our SA algorithm. Indeed, it is able to improve on 2 previous best-known solutions and to equal these results in 13 instances. For the other 3 tested instances, SA did not reach the best reported solutions but its results are very close to them ($\Delta \leq 0.002$).

On the other hand, we can observe that the results produced by ILS are better than those furnished by DA since it is able to match 6 best-known solutions and DA only 2. For the rest of the instances these algorithms did not reach the best reported results. Thus, we can conclude from this experiment that SA is a better metaheuristic algorithm than DA and ILS for solving the selected benchmark instances of the MP problem. This can be better observed in Fig. 6, which presents for each studied instance a performance comparison between these metaheuristic algorithms and DA with respect to the best-known solutions.

Instance	DA	Δ	ILS	Δ	SA	Δ	Best-known
ANGI	216	0.000	216	0.000	216	0.000	216
CARP	564	0.029	549	0.002	548	0.000	548
ETHE	374	0.005	373	0.003	372	0.000	372
GOLO	503	0.014	496	0.000	496	0.000	496
GRIS	172	0.000	172	0.000	172	0.000	172
ROPA	329	0.012	325	0.000	325	0.000	325
SCHU	771	0.016	759	0.000	759	0.000	759
TENU	685	0.004	682	0.000	682	0.000	682
tst01	558	0.024	550	0.009	545	0.000	545
tst02	1370	0.010	1364	0.006	1355	-0.001	1356
tst03	851	0.022	844	0.013	833	0.000	833
tst04	602	0.024	592	0.007	588	0.000	588
tst05	805	0.020	796	0.009	789	0.000	789
tst06	612	0.027	606	0.017	596	0.000	596
tst07	1302	0.025	1290	0.016	1271	0.001	1270
tst08	876	0.027	871	0.021	855	0.002	853
tst09	1168	0.020	1157	0.010	1146	0.001	1145
tst10	736	0.021	731	0.014	720	-0.001	721
Average		0.0167		0.0071		0.0001	

Table 11. Performance comparison between two proposed metaheuristics: ILS and SA.

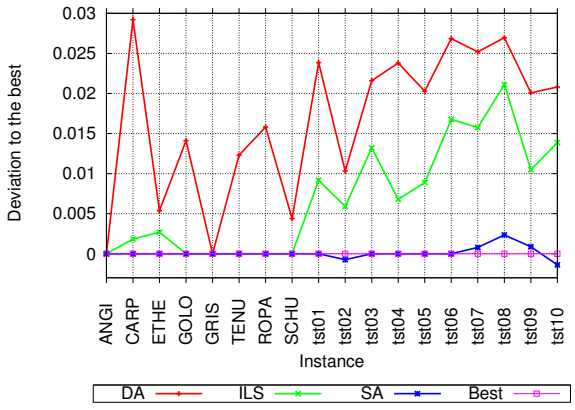

Figure 6. Performance comparison between two metaheuristic algorithms and DA for the MP problem.

Regarding the computational time expended by the analyzed methods we have carried out an additional experimental comparison. The results of this comparison are summarized in Table 12. It presents for each algorithm the best solution achieved (Cost) in one execution over the instance *tst08*, the expended computational time in seconds, and the total number of iterations needed, i.e., the total calls to the objective function $\phi(T)$.

Algorithm	Cost	Time	Iterations
DA	891	0.15	1206
ILS	881	97.36	1365220
SA	867	27.00	357131

Table 12. Computational time comparison between two metaheuristic algorithms and DA over the instance *tst08*.

From Table 12 one observes that DA has the best running time with 0.15 seconds, but its cost of 891 was the worst. The ILS algorithm provided a better cost than DA (881), however, it consumes considerably much more computational time than any of the analyzed algorithms (97.36 seconds). The best cost was obtained by the SA algorithm (867) by expending only 27.00 seconds.

The computational effort of the compared algorithms can be better appreciated in Fig. 7. It plots three execution profiles which represent the evolution of the best solutions attained by the compared algorithms over the instance *tst08*. From this figure we can clearly observe that the SA algorithm provides the best trade-off between the needs for good quality solutions, and reasonable consumed computational time. This figure allows us to summarize the overall behavior of the compared algorithms since similar results were obtained with all the other tested instances.

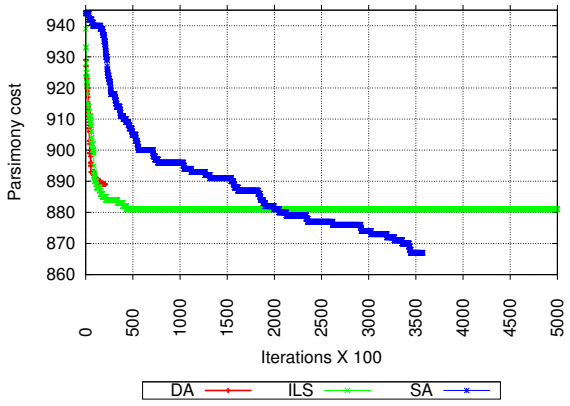

Figure 7. Execution profiles produced during one run of the compared algorithms over the instance *tst08*.

5 Conclusion

In this paper we have presented two different metaheuristic methods for finding near-optimal solutions for the MP problem under the Fitch's criterion, an iterated local search (ILS) and a simulated annealing (SA). Different possibilities for the key components of these algorithms were carefully designed and analyzed in order to discover the combination of them which yields the best quality solutions to the problem at a reasonable computational effort.

The performance of both metaheuristics was investigated through extensive experimentation over a set of 18 well known benchmark instances. The results from this experiments have shown that our SA algorithm is able to improve 2 best-known solutions (*tst02* and *tst10*) and to match these results in 13 instances. For the other 3 tested instances, SA did not reach the best reported solutions but its results present a very small deviation with respect to them ($\Delta \leq 0.002$).

Finding near-optimal solutions for the MP problem is a very challenging problem. However, the introduction of this SA algorithm 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.

Regarding our ILS implementation, we consider that is necessary to test different perturbation methods and acceptation criteria.

6 Acknowledgements

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

References

[1] W. Hennig, *Phylogenetic Systematic*. Phylogeny, University of Illinois, 1966.

[2] D. M. Hillis, C. Moritz, B. K. Mable, and R. G. Olmstead, *Molecular systematics*, vol. 23. Sinauer Associates Sunderland, MA, 1996.

[3] W. M. Fitch and E. Margoliash, "A method for estimating the number of invariant amino acid coding positions in a gene using cytochrome c as a model case," *Biochemical Genetics*, vol. 1, no. 1, pp. 65–71, 1967.

[4] N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees," *Molecular Biology and Evolution*, vol. 4, no. 4, pp. 406–425, 1987.

[5] J. Felsenstein, "Evolutionary trees from dna sequences: a maximum likelihood approach," *Journal of molecular evolution*, vol. 17, no. 6, pp. 368–376, 1981.

[6] A. W. F. Edwards and C. L. L. Sforza, "The reconstruction of evolution," *Heredity*, vol. 18, 1963.

[7] L. L. Cavalli-Sforza and A. W. F. Edwards, "Phylogenetic analysis. models and estimation procedures," *American Journal of Human Genetics*, vol. 19, no. 3 Pt 1, pp. 233–257, 1967.

[8] D. Penny, L. R. Foulds, and M. D. Hendy, "Testing the theory of evolution by comparing phylogenetic trees constructed from five different protein sequences," *Nature*, vol. 297, pp. 197–200, 1982.

[9] E. Sober, *The nature of selection*. The MIT Prs., 1987.

[10] D. M. Gusfield, *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. University of Cambridge, 1st. ed., 1997.

[11] M. Garey and D. Johnson, "The rectilinear steiner tree problem is np-complete," *SIAM Journal on Applied Mathematics*, vol. 32, no. 4, pp. 826–834, 1977.

[12] J. Xiong, *Essential Bioinformatics*. Cambridge University Press, 1st. ed., 2006.

[13] M. D. Hendy and D. Penny, "Branch and bound algorithms to determine minimal evolutionary trees," *Mathematical Biosciences*, vol. 59, no. 2, pp. 277–290, 1982.

[14] A. Andreatta and C. C. Ribeiro, "Heuristics for the phylogeny problem," *Journal of Heuristics*, vol. 8, no. 4, pp. 429–447, 2002.

[15] C. C. Ribeiro and D. S. Vianna, "A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure," *International Transactions in Operational Research*, vol. 12, no. 3, pp. 325–338, 2005.

[16] H. Matsuda, "Protein phylogenetic inference using maximum likelihood with a genetic algorithm," *Proceedings of 1st Pacific Symposium on Biocomputing*, vol. 3, no. 1, pp. 512–523, 1996.

[17] P. O. Lewis, "A genetic algorithm for maximum likelihood phylogeny inference using nucleotide sequence data," *Molecular Biology and Evolution*, vol. 15, no. 3, pp. 277–283, 1998.

[18] J. M. Richer, A. Goëffon, and J. K. Hao, "A memetic algorithm for phylogenetic reconstruction with maximum parsimony.," in *EvoBIO*, vol. 5483 of *Lecture Notes in Computer Science*, pp. 164–175, Springer, 2009.

[19] O. Martin, S. W. Otto, and E. W. Felten, "Large-step Markov chains for the traveling salesman problem," *Complex Systems*, vol. 5, no. 3, pp. 299–326, 1991.

[20] H. R. Lourenco, O. Martin, and T. Stützle, "Iterated local search," in *Handbook of Metaheuristics*, vol. 57 of *International Series in Operations Research & Management Science*, pp. 320–353, Kluwer Academic Publishers, 2003.

[21] M. S. Waterman and T. F. Smith, "On the similarity of dendrograms," *Journal of Theoretical Biology*, vol. 73, no. 4, pp. 784–900, 1978.

[22] D. Robinson, "Comparison of labeled trees with valency three," *Journal of Combinatorial Theory, Series B*, vol. 11, no. 2, pp. 105–119, 1971.

[23] D. L. Swofford and G. J. Olsen, "Phylogeny reconstruction," in *Molecular Systematics*, pp. 411–501, USA: Sinauer Associates, Inc., Sunderland, 1990.

[24] B. Allen and M. Steel, "Subtree transfer operations and their induced metrics on evolutionary trees," *Annals of Combinatorics*, vol. 5, no. 1, pp. 1–15, 2001.

[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[26] V. Cerny, "A thermodynamic approach to the traveling salesman problem: An efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, no. 1, pp. 41–51, 1985.

[27] E. G. Talbi, *Metaheuristics: from design to implementation*. Hoboken, New Jersey: John Wiley & Sons, 1st ed., 2009.

[28] P. Goloboff, "Nona, version 1.6," *Published by the author*, 1997.

[29] M. Luckow and R. A. Pimentel, "An empirical comparison of numerical wagner computer programs," *Cladistics*, vol. 1, no. 1, pp. 47–66, 1985.

[30] N. I. Platnick, "An empirical comparison of microcomputer parsimony programs, II," *Cladistics*, vol. 5, no. 2, pp. 145–162, 1989.

[31] A. Goëffon, *Nouvelles Heuristiques de Voisinage et Mémétiques pour le Probléme Maximum de Parcimonie*. PhD thesis, Université D'Angers, november 2006.

[32] C. C. Ribeiro and D. S. Vianna, "A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking," *Proc. of 2nd Bresil Workshop on Bioinformatics*, pp. 97–102, 2003.