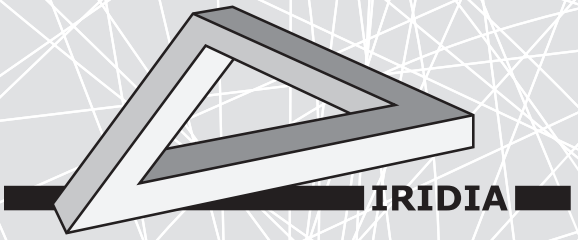


**Université Libre de Bruxelles**

*Institut de Recherches Interdisciplinaires*
*et de Développements en Intelligence Artificielle*

# Metaheuristic Approaches for Inferring Phylogenies

Daniele CATANZARO

**Technical Report N°**

TR/IRIDIA/2004-2

March 2004

Université Libre de Bruxelles

# Metaheuristic Approaches for Inferring Phylogenies

by

**Daniele Catanzaro**

IRIDIA, Université Libre de Bruxelles
Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium
dcatanzaro@iridia.ulb.ac.be

Supervised by

**Marco Dorigo, Ph.D.**

IRIDIA, Université Libre de Bruxelles
Avenue Franklin Roosevelt 50, CP 194/6, 1050 Brussels, Belgium
mdorigo@ulb.ac.be

and

**Michel Milinkovitch, Ph.D.**

UEG, Université Libre de Bruxelles
IBMM, CP300, Rue Jeener & Brachet, B-6041, Gosselies, Belgium
mcmilink@ulb.ac.be

May, 2003

Memoire présenté en vue de l'obtention du DIPLOME D'ETUDES APPROFONDIES (DEA) en sciences appliquées

*A mio fratello Simone.*

# Contents

# List of Figures

*Nomen dei in litterarum permutatione est*

# Chapter 1

# Phylogenetic analysis

## 1.1 Introduction

Phylogenetics is the study of evolutionary relationships. It deals with finding descent relationships in terms of adaptive modifications based on natural and sexual selection.

The means of inferring or estimating these evolutionary relationships is called phylogenetic analysis. The result of this inference is depicted using tree-like diagrams—called phylogenetic trees—representing estimated lineages of the inherited relationships among molecules, organisms, or both. Phylogenetic trees are fundamental to understand the evolution and the diversity, necessary to organize biological data and central for the organisms comparison.

We can distinguish among *classic* and *modern* phylogenetics. Classic phylogenetics deals with physical or morphological features—called also characters—of a set of entities, for example: size, color, number of legs and so on. This approach is also called *cladistics* because clade, a set of descendants from a single ancestor, is a Greek term meaning branch. Cladistics supposes that members of a group or clade share a common evolutionary history and are more related to each other than to members of another group. A given group is recognized by sharing unique features that were not present in distant ancestors. These shared and derived characteristics can be anything that can be observed and described.

Modern phylogenetics, at the contrary, uses information extracted from genetic material—mainly DNA and protein sequences. The stable inheritance of characteristics in fact is mediated by the genome so the relationships between species can be deduced from their genomic information.

The Classic and the modern approach however have in common the goal:

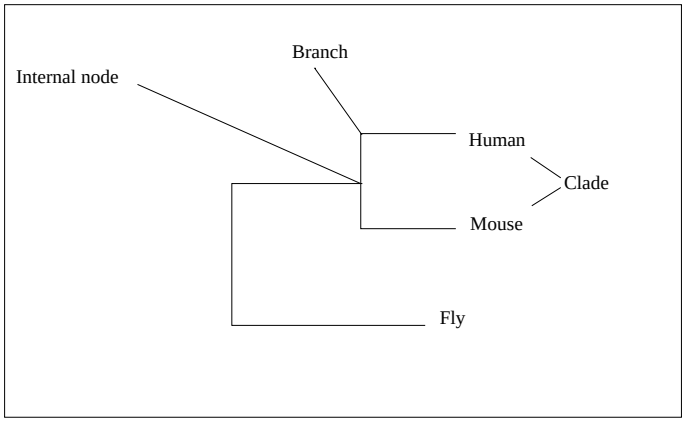


Figure 1.1: Trivial example of phylogenetic tree.

to obtain the phylogenetic tree of a given set of species. The next section will define the basic terms of this representation.

## 1.2 Definitions of terms and number of topologies

By observing the tree represented in Figure 1.1, we can identify the basic components of a phylogenetic tree:

**internal node:** a bifurcating branch point within the tree or, more simply, a not peripheral node (leaf).

**branch:** a divergence between two nodes; it is usually represented as a line. In Figure 1.1 for example, the link between the internal node and the human in a branch.

**branch length:** the evolutionary distance between two nodes or, in other words, the number of evolutionary changes necessary to evolve from a node A to a node B.

**taxon:** any named group of organisms; it is always a peripheral node and can be also called **external node**.

**clade:** a group of organisms sharing a common ancestor.

Usually phylogenetics represents its diagrams depicting *un-rooted* and *bifurcating* (binary) trees. Un-rooted because a typical assumption for the

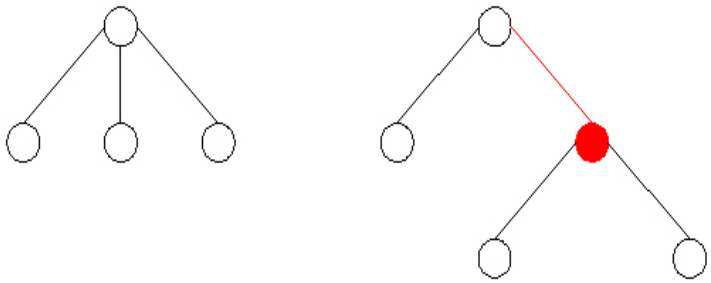

Figure 1.2: As we can see in this example, a ternary tree can be reduced to a binary tree adding a dummy node and a dummy edge.

tree building is the reversibility of the model:"a evolved to b" or "b evolved to a" must be equiprobable events [Swofford et al., 1996]. This assumption imposes that the tree must be un-rooted because the presence of a root in a tree would presuppose an initial starting point— that is a direction in the acyclic graph— and this would contradict the hypothesis of reversibility.

About the second attribute, considering binary tree is not reductive: it is possible to demonstrate that a n-ary tree can be reduced to a binary tree adding dummy nodes and dummy edges as in Figure 1.2. Furthermore, if the tree is binary, we have equations that allow us to quantify the number of internal nodes, the number of the branches and the number of tree topologies for an assigned number of taxa [Swofford et al., 1996, Felsenstein, 2002]: considering $T$ external nodes, an un-rooted and binary tree is characterized by $T-2$ internal nodes, $2T-3$ branches of which $T-3$ link internal nodes and $T$ link leaves; the total number of distinct un-rooted binary trees is

$$B(T) = \prod_{i=3}^{T} (2i-5) \sim 2^T \cdot T. \tag{1.1}$$

Adding a root adds one more internal node and one more interior branch; since the root can be placed along any of the 2T-3 branches, the number of possible rooted trees is increased by a factor of 2T-3. Equation 1.1 means that the number of tree topologies for a given set of taxa is exponential varying the number of taxa involved in the analysis, therefore the explicit enumeration of the trees is intractable even for small values of $T$ (greater than 13).

## 1.3 Tree building methods

The intractability of the problem leads to the birth of a set of methodologies and techniques finalized to finding a good approximation of the best phylogenetic tree for a given set of taxa [Baxevanis and Ouellette, 2001, Steel et al., 1998]. Considering the type of data used for reaching this objective we can distinguish among character-based methods and distance-based methods. The former use *characters*, the latter use *distances*. From now on, we will use as synonymous the term species and the DNA sequence that describes it.

### 1.3.1 Character-based methods

A discrete character provides data about a given sequence. For example, given a DNA sequence, a base in a specific position of the sequence represents a discrete character.

...AATTGCATGATGGGGCCCTATTTGGAAAA...

A discrete character $x$ can assume values in the set of character states

$$S = S_1 \cup S_2 \tag{1.2}$$

where $S_1$=[A, T, C, G] and $S_2$=[M, R, W, S, Y, K, V, H, D, B, N].

$S_1$ is made up by the four bases [1] Adenine (A), Timine (T), Citosine (C), Guanine (G) while $S_2$ is made up by literals belonging to a table called *standard ambiguity codes* (SAC). SAC has reason to exist because sometimes we don't know anything about the base in exam or we have very poor information (e.g. the case of corrupted sequences, Swofford et al. [1996]). The standard ambiguity code is represented in Figure 1.3.

We can define therefore a discrete character as an independent variable whose possible values are collections of mutually exclusive character states [Shamir and Naor, 2002]. The assumption of independence among characters is common to most character-based methods of analysis. When we can not assume independence, we are forced to take covariances among characters into account, and the computational methods become considerably more complicated. Furthermore, the assumption of independence enables us to

[1] Adenine and Timine are usually called purine while Citosine and Guanine are called pyramidine.

| code | description |
|---|---|
| M | $A \vee C$ |
| R | $A \vee G$ |
| W | $A \vee T$ |
| S | $C \vee G$ |
| Y | $C \vee T$ |
| K | $G \vee T$ |
| V | $A \vee C \vee G$ |
| H | $A \vee C \vee T$ |
| D | $A \vee G \vee T$ |
| B | $C \vee G \vee T$ |
| N | $A \vee C \vee G \vee T$ |

Figure 1.3: The standard ambiguity code.

treat each position separately, thereby allowing problems to be subdivided into a number of much simpler subproblems (divide et impera approach).

A second assumption is the following: characters must be homologous. By homology we mean that a character must be defined in such a way that all of the states observed over taxa for that particular character must have been derived, perhaps with modification, from a corresponding state observed in the common ancestor of those taxa.

### Maximum parsimony

Maximum parsimony (MP) is an optimization criterion that adheres to the principle that the best explanation of the data is the simplest, which in turn is the one requiring the fewest ad hoc assumptions. An example of the maximum parsimony method can be found in Figure 1.4. In practical terms, the MP tree is the shortest, that is the one with the fewest changes. There are several variants of MP that differ with regard to the permitted directionality of character state change [Swofford et al., 1996]. To accommodate substitution bias, MP is amenable to weighting; for example, the transformation of a transversion[2] can be weighted relative to a transition [3]. The easiest way

[2]We define *transversion* one of the eight types of substitution: $A \longrightarrow C$, $A \longrightarrow T$, $C \longrightarrow G$, $G \longrightarrow T$, and the reverse.

[3]We define transition one of the four types of substitution $A \longrightarrow G$, $G \longrightarrow A$, $C \longrightarrow T$, $T \longrightarrow C$. Transitions are usually more frequent than transversion. MP wants to minimize the number of transitions and transversions necessary for explaining a given tree.

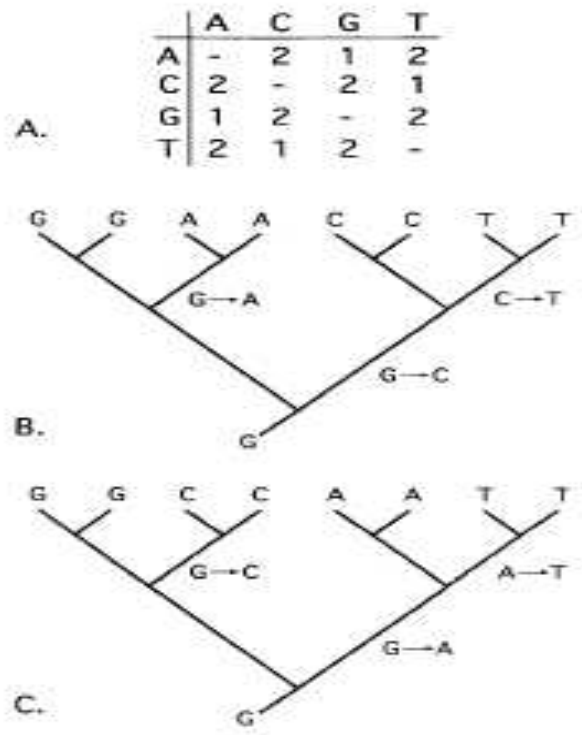


Figure 1.4: Character weight matrix and application in MP phylogenetic analysis. (A) Matrix indicating that a transversion substitution costs twice that of a transition. Because, according to MP, bases shared between two sequences cannot ever have changed, diagonal elements of the matrix are ignored. (B, C) Two phylogenetic resolutions and reconstructions of the evolution of a hypothetical pattern of aligned bases at a particular site in eight sequences. With un-weighted MP, both reconstructions (among several others) have the same cost (three steps); hence, they are equally acceptable. With the weight matrix in (A), the reconstruction of (B) requires four steps, and the reconstruction of (C) requires five. Thus, the first reconstruction (B) and others requiring four steps are preferred.

to do this is to create a weighting step matrix in which the weights are the reciprocal of the rates estimated using maximum likelihood (ML) (described subsequently). However, step-matrix weighting can greatly slow MP computation [Huelsenbeck, 1995]. MP analysis tend to yield numerous trees that have a same score. Because each is defined to be as optimal as any other, only groupings present in the strict *consensus*[4] of all trees are considered to be supported by the data (for more information see [Swofford et al., 1996, Felsenstein, 2002]).

Simulation studies have shown that MP performs no better than minimum evolution (ME) and worse than ML (described subsequently) when the amount of sequence evolution since lineages diverged is much greater than the amount of divergence that occurred between lineage splits (i.e., in a tree with very long terminal branches and short internal internodes) [Huelsenbeck, 1995]. This condition produces "long branch attraction"—the long branches become artificially connected because the number of non-homologous similarities the sequences have accumulated exceeds the number of homologous similarities they have retained with their true closest relatives [Swofford et al., 1996]. Character weighting improves the performance of MP under these conditions [Huelsenbeck, 1995]. We will see better MP calculation in the next chapter.

### Maximum likelihood

Maximum likelihood (ML) turns the phylogenetic problem inside out. ML searches for the evolutionary model that has the highest likelihood of producing the observed data. In practice, ML is derived for each base position in an alignment. The likelihood is calculated in terms of the probability that the pattern of variation at a site would be produced by a particular substitution process, given a particular tree and the overall observed base frequencies. The likelihood becomes the sum of the probabilities of each possible reconstruction of substitutions under a particular substitution process. The likelihoods for all the sites are multiplied to give an overall "likelihood of the tree" (i.e., the probability of the data given the tree and the substitution process). As one can imagine, for one particular tree, the likelihood of the data is low at some sites and high at others. For a "good" tree, many sites will have high likelihood, so that the product of likelihoods is high. For a "poor" tree, the reverse will be true.

The substitution model should be optimized to fit the observed data.

[4]Basically the consensus tree consists of groups that occur as often as possible in the data. If a group occurs in more than 50% of all the input trees it will definitely appear in the output tree (called consensus tree).

Because ML uses great amounts of computational time, it is usually impractical to perform a complete search that simultaneously optimizes the substitution model and the tree for a given data set. Economical heuristic approaches are recommended [Adachi and Hasegawa, 1992, Swofford et al., 1996, Stamatakis and Ludwig, 2003] and [Stamatakis and Ludwig, 2002, Stamatakis et al., 2004a,b]. Perhaps the best time saver in this regard is preliminary ML estimation of the substitution model (as can be performed using PAUP( http://paup.csit.fsu.edu)). This procedure can be applied iteratively, searching for better ML trees, then re-estimating the parameters, and then searching for better trees. As algorithms, computers, and phylogenetic understanding have improved, the ML criterion has become more popular for molecular phylogenetic analysis. In simulation studies, ML has consistently outperformed ME and MP when the data analysis proceeds according to the same model that generates the data [Huelsenbeck, 1995]. ML will always be the most computationally intensive method of all, however, so there will always be situations in which it is not practical.

### 1.3.2 Distance-based methods

Distance-based methods use the amount of dissimilarity (the distance) between two aligned sequences to derive trees. A distance method would reconstruct the true tree if all genetic divergence events were accurately recorded in the sequence. However, divergence encounters an upper limit as sequences become mutationally saturated [Felsenstein, 2002]. After one sequence of a diverging pair has mutated at a particular site, subsequent mutations in either sequence cannot render the sites any more "different". In fact, subsequent mutations can make them again equal [Baxevanis and Ouellette, 2001, Shamir and Naor, 2002, Felsenstein, 2002]. Therefore, most distance-based methods correct for such "unseen" substitutions.

Pairwise distance is calculated using ML estimators of substitution rates. The most popular distance tree-building programs have a limited number of substitution models, but PAUP 4.0 implements a number of models, including the actual model estimated from the data using ML, as well as the logdet distance method. Distance methods are much less computationally intensive than maximum likelihood but can employ the same models of sequence evolution. This is their biggest advantage. The disadvantage is that the character data are discarded and with them all local information contained inside them. The most commonly applied distance-based methods are the un-weighted pair group method with arithmetic mean (UPGMA), neighbor joining (NJ), and methods that optimize the additivity of a distance tree, including the minimum evolution (ME) method [Baxevanis and Ouellette,

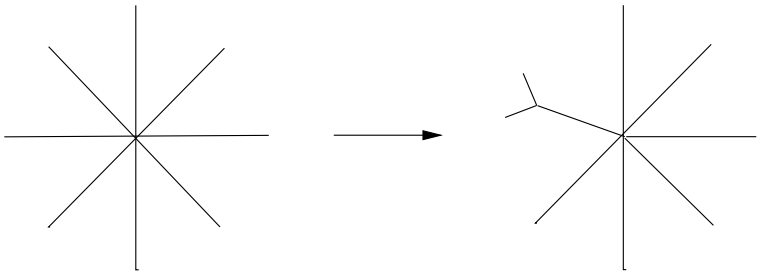

Figure 1.5: Star decomposition. This is how tree-building algorithms such as neighbor joining work. The most similar terminals are joined, and a branch is inserted between them and the remainder of the star. Subsequently, the new branch is consolidated so that its value is a mean of the two original values, yielding a star tree with n-1 terminals. The process is repeated until only one terminal remains.

2001, Felsenstein, 2002, Shamir and Naor, 2002].

### Un-weighted pair group method with arithmetic mean

Un-weighted Pair Group Method with Arithmetic Mean (UPGMA) is a clustering algorithm; it joins tree branches based on the criterion of greatest similarity among pairs and averages of joined pairs [Shamir and Naor, 2002]. UPGMA is expected to generate an accurate topology with true branch lengths only when the divergence is according to a molecular clock [Swofford et al., 1996] or approximately equal to raw sequence dissimilarity. These conditions are rarely met in practice.

### Neighbor joining

The neighbor-joining algorithm (NJ) is commonly applied with distance tree building, regardless of the optimization criterion. The fully resolved tree is "decomposed" from a fully unresolved "star" tree by successively inserting branches between a pair of closest (actually, most isolated) neighbors and the remaining terminals in the tree Figure 1.5. The closest neighbor pair is then consolidated, effectively reforming a star tree, and the process is repeated. The method is comparatively rapid [Shamir and Naor, 2002].

### Minimum evolution

Minimum evolution (ME) seeks to find the shortest tree that is consistent with the path lengths measured by minimizing the squared deviation of all

possible distance relative to all possible path length on the tree [Rzhetsky and Nei, 1992, Swofford et al., 1996, Felsenstein, 2002]. ME does not use all possible pairwise distances and all possible associated tree path lengths, rather, it fixes the location of internal tree nodes based on the distance to external nodes and then optimizes the internal branch length according to the minimum measured error between these "observed" points.

## Existing phylogenetics software

PHYLIP and PAUP are the most widely used phylogenetic analysis software, although other newer applications such as PUZZLE are beginning to compete. Here, PHYLIP and PAUP will be described in the most detail, with references made to other available packages that have useful features. However, the number of programs available is now so numerous, many having their own useful features, that the reader is referred to the list of Internet resources in [Baxevanis and Ouellette, 2001] for further information.

### PHYLIP

PHYLIP (phylogeny inference package) is a package consisting of about 30 programs that cover most aspects of phylogenetic analysis. PHYLIP is free and available for a wide variety of computer platforms (Mac, DOS, UNIX, VAX/VMS, and others). According to its author, PHYLIP is currently the most widely used phylogeny program. PHYLIP is a command-line program and does not have a point-and-click interface, as programs like PAUP do. The tree file generated is a widely used format that can be imported into a variety of tree-drawing programs, including DRAWGRAM and DRAWTREE that come with this package. However, these PHYLIP tree-drawing programs produce low-resolution graphics, so a program such as TreeView (described below) is instead recommended. Particulars of some of the PHYLIP tree-inference programs are discussed below. PROTDIST is a program that computes a distance matrix for an alignment of protein sequences. It allows the user to choose between one of three evolutionary models of amino acid replacements. NEIGHBOR is a tree-generating program that utilizes the distance matrix data generated from a program such as PROTDIST and generates a tree using the neighborjoining method. This is one of the more popular methods, due to its speed of computation. FITCH is another tree-generating program similar to NEIGHBOR but much more robust. It also uses distance matrix data, such as that described in PROTDIST, and generates a tree using the method of Fitch-Margoliash. This method, while

more robust than NEIGHBOR, tends to produce a similar final answer, yet takes longer to compute. Although computational times are often significantly longer, the quality of the results produced by the method often makes this method the method of choice in these types of analyses. PROTPARS is a parsimony program for protein sequences that generates trees without utilizing a distance matrix. The evolutionary model is different from the ones used in the PROTDIST program in that it considers the underlying changes in the nucleotide sequence to evaluate the probabilities of the observed amino acid changes. PROTPARS does not have an option that uses empirical values for amino acid changes (e.g., PAM matrices). DNADIST computes a distance matrix from nucleotide sequences. Trees are generated by running the output through NEIGHBOR or other distance matrix programs in the PHYLIP package. DNADIST allows the user to choose between three models of nucleotide substitution. The older Jukes and Cantor model is similar to the simple model in the PROTDIST program in that it assumes equal probabilities for all changes. The more recent Kimura two-parameter model is very similar but allows the user to weigh transversion more heavily than transitions. PHYLIP also comprises DNAML, a maximum-likelihood program for nucleotide data. Because the program is fairly slow, the use of its faster "sibling", the fastDNAml program [Olsen et al., 1994] described below, is recommended. SEQBOOT and CONSENSE are required for bootstrap analysis. SEQBOOT is used to generate any number of replicates of the data; these replicates are then used in programs within the PHYLIP suite for analysis. The resulting tree file contains as many trees as there are replicates of the data, so this file needs to be run through CONSENSE to generate the consensus tree from the analysis.

**PAUP**

The objective of the development of PAUP is to provide a phylogenetics program that includes as many functions (including tree graphics) as possible in a single, platform-independent program with a menu interface. PAUP stands for phylogenetic analysis using parsimony and contains one of the most sophisticated parsimony programs available. Current tree-building functions in PAUP include MP, and, for nucleotide data, distance and ML using the fastDNAml algorithm. Each tree-building program permits a variety of options. The MP options include specification of any character-weighting scheme. Distance options include choice of NJ, ME, FM (see PAUP release notes regarding PHYLIP), and UPGMA procedures. According to the release notes accompanying PAUP test version 4, PAUP* usually finds trees with likelihoods as high or higher [i.e., better] than PHYLIP (both because PAUP's tree

rearrangements are more extensive and because its convergence criterion for branch-length iteration is stricter). With any tree-building method, PAUP allows a variety of tree search options. These include algorithm specification for generating the initial tree (starting tree): NJ, stepwise addition, or input tree(s). For MP analysis, in which a large number of equal-length trees might exist, the search should specify saving from each replicate only a few trees that match or are better than the score of the slower search. PAUP performs the Kishino-Hasegawa test to compare MP or ML trees, computes four types of consensus of multiple trees (usually used for multiple equal-length MP trees), computes stepwise differences between MP trees, and evaluates signal conflicts between specified partitions of sites (e.g., nuclear and organical sequence data in a combined analysis).

### 1.3.3 Other programs

In addition to PAUP and PHYLIP, there are phylogenetics programs that have some unique capabilities but are generally more limited in their procedures and portability. These include FastDNAml, PUZZLE, and MOLPHY.

#### FastDNAml

FastDNAml [Olsen et al., 1994] is a freestanding maximum-likelihood, tree-building program. Although it is currently not part of the PHYLIP package, it uses largely the same input and output conventions, and the results of fastDNAml and PHYLIP's DNAML should be very similar or identical. FastDNAml can be run on parallel processors, and it comes with a number of useful scripts (in particular for bootstrapping and jumbling the sequence input order). To take full advantage of the program, knowledge of UNIX is beneficial. The source code for UNIX systems is publicly available from the RDP Web site, and a Power Macintosh version is available by FTP.

#### PUZZLE

PUZZLE or TREE-PUZZLE [Strimmer and von Haeseler, 1996], as it is now called, is a maximum likelihood-based program that implements a fast tree search algorithm (quartet puzzling) that allows analysis of large data sets and automatically assigns estimations of support to each internal branch. PUZZLE also computes pairwise maximum-likelihood distances as well as branch lengths for user-specified trees. PUZZLE also offers a novel method, likelihood mapping, to investigate the support of a hypothesized internal branch without computing an overall tree and to visualize the phylogenetic

content of a sequence alignment. It conducts a number of statistical tests ( $\chi^2$ test for homogeneity of base composition, likelihood ratio clock test, Kishino-Hasegawa test) and includes a large range of models of substitution. Rate heterogeneity is modeled by a discrete gamma distribution and by allowing invariable sites.

### MOLPHY

MOLPHY is a shareware package of programs and utilities for ML analysis and statistics of nucleotide or amino acid sequences [Adachi and Hasegawa, 1992]. It has been tested on Sun OS and HP9000/700 systems. Practical application requires some knowledge of UNIX file management. The ML procedures are similar to those in PHYLIP, but there is a wider range of amino acid substitution models and options for faster, heuristic searches, including an option to use "local bootstrap" analyses (i.e., a bootstrap on subtrees under the assumption that the remainder of the tree is correct) to search for better ML trees. The output includes branch-length estimates and standard error. Analysis of separate codon positions is possible. MOLPHY uses a subset of the nucleotide substitution models available in PAUP, although it allows user-specified parameter values. The current MOLPHY lacks a bootstrap option and also has no accommodation for among-site rate heterogeneity.

### METAPIGA

MetaPIGA (version 1.02) is written in the cross-platform compatible (Windows, Macintosh, Unix/Linux) Java programming language and implements a graphical interface for importing/exporting data, specifying a ML model, monitoring search progress and visualizing trees generated by the engine. The current version allows the analysis of data sets of 500 taxa and 3000 nucleotides on a regular computer (1.7-Gh Pentium IV) in less than 10 hours under the Hasegawa-Kishino-Yano (HKY) + rate heterogeneity model. As in [Agarwala and Fernández-Baca, 1994] and [Reijmers et al., 1999], the core of MetaPIGA is based on a genetic algorithm, and makes a strong use of the concept of consensus pruning (see http://www.ulb.ac.be/sciences/ueg/html-files/MetaPIGA.html for further information).

# Chapter 2

# Modeling the problem

In this chapter we will model the phylogenetic inference problem, one of the most important problem in the molecular biology field, and we will show that it is a combinatorial optimization problem.

The definition of solution for the problem, and space of solutions of the problem can be found in the first section of this chapter. Subsequently, in the second section, we describe in which way we model a solution, and in the third section we give the definitions of neighborhood of a solution, defining the operators that let obtain a neighborhood starting from a given solution. The fourth section introduces the quality function used for evaluating a solution, and describes in which way to compute it.

## 2.1 Problem definition

Let $V = E \cup I$ be the set of nodes and $B$ the set of the edges of a tree[1], where $E$ is the set of leaves[2] and $I$ is the set of internal nodes.

A tree can be represented using an adjacency matrix $M$ in which each $m_{ij}$ is equal to 1 if there exists a link between nodes $v_i$ and $v_j$, 0 otherwise [Rosen, 2000]; $M$ explicitly maps the tree topology, so we will say that two trees $T_1$ and $T_2$ are topologically equivalent if $m_{ij_{T_1}} = m_{ij_{T_2}} \forall\ i$ and $j$, where $m_{ij_{T_1}}$ is the generic element of the matrix $M_{T_1}$, respectively for $m_{ij_{T_2}}$. Otherwise, we will say that $T_1$ and $T_2$ are topologically different.

As we showed in chapter 1, given a set $\Gamma$ of taxa such that $\mid \Gamma \mid = n$, the number of possible distinct tree topologies is $2^n \cdot n$. This means that for each

[1] We use here the classical definition of undirect tree: an acyclic, undirect and connected graph G=(V,B).

[2] From now on we will use as synonymous "leaves", "set of taxa" and "set of external nodes".

given set $\Gamma$, there exists a set $S$ made of $2^n \cdot n$ distinct adjacency matrices representing these topologies.

We can formally define the phylogenetic inference problem like a search problem in the following way[3]:

**Problem 2.1** *(Search problem)*

$$find \quad M \tag{2.1}$$

$$s.t. \sum_{j=1}^{2n-1} m_{ij} \leq 3 \quad i \in I \tag{2.2}$$

$$\sum_{j=1}^{2n-1} m_{ij} \geq 2 \quad i \in I \tag{2.3}$$

$$\sum_{j \in I} m_{ij} = 1 \quad i \in E \tag{2.4}$$

$$\sum_{j \in E} m_{ij} = 0 \quad i \in E \tag{2.5}$$

The constraint 2.2 means that each internal node can have at most three links with the other nodes of the tree, this imposes that the tree must be binary. The constraint 2.3 imposes that each internal node must have no less that two links, this means that an internal node cannot be a leaf. The constraint 2.4 imposes that each external node must have exactly a link with the other nodes of the tree, this means that an external node cannot be an internal node, and the latter constraint 2.5 imposes that no link can exist between leaves.

Mathematical solutions for the search problem are all the matrices that satisfy its constraints. Not all these solutions are empirically valid: for example, if we consider Figure 2.1, we can observe that the phylogeny represented can not be valid because a man and a monkey are two species nearer than a man and a dolphin or a monkey and a dolphin. This problem can be avoided introducing a function that, embodying the parameters for judging the goodness of a solution, distinguishes a solution empirically valid from another one that is not.

If we assume that there exists a function $f : S \rightarrow \mathbb{R}$ that assigns to each solution $M$ in $S$ a real number, we can give a formal definition of the phylogenetic inference problem as an optimization problem:

[3]As we have seen in chapter 1, the number of internal node for a binary tree (also called fully resolved tree) is $\mid \Gamma \mid -1$ where $\Gamma$ is the set of taxa and $\mid \Gamma \mid$ is its cardinality. From now on we will always suppose $\mid \Gamma \mid = n$.

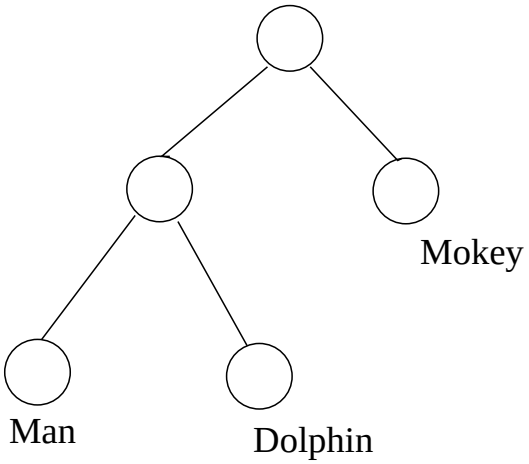


Figure 2.1: An example of mathematically correct solution for the search problem, but refutable empirically: man and monkey are nearer than man and dolphin.

**Problem 2.2** *(Optimal phylogenetic tree)*

$$minimize \quad f(M) \tag{2.6}$$

$$s.t. \sum_{j=1}^{2n-1} m_{ij} \leq 3 \quad i \in I \tag{2.7}$$

$$\sum_{j=1}^{2n-1} m_{ij} \geq 2 \quad i \in I \tag{2.8}$$

$$\sum_{j \in I} m_{ij} = 1 \quad i \in E \tag{2.9}$$

$$\sum_{j \in E} m_{ij} = 0 \quad i \in E \tag{2.10}$$

Hypothesizing to have a set of $m$ characters relative to each leaf $v$ of the tree $M$, calling $v_j$ the $j$-th character of $v$, $A(M)$ the set of branches, hypothesizing to have a set of $k$ states for each character, and defining *value* of the tree $M$ the number:

$$f(M) \equiv \sum_{(u,v) \in A(M)} |\{j : v_j \neq u_j\}| \tag{2.11}$$

that is, the total number of times that the state of some character changes along some edge, it is possible to demonstrate (see [Shamir and Naor, 2002,

Gusfield, 1984]) that the optimization problem 2.2 is NP-hard. In particular, it can be shown (see [Gusfield, 1984]) that, given a matrix $M$ describing $m$ characters of a set of $n$ species, to find a most parsimonious tree, that is a tree[4] such that the function 2.11 is minimal, is reducible to the un-weighted Steiner tree problem on an m-dimensional hypercube. The latter is a well-known NP-hard problem [Papadimitriou and Steiglitz, 1998, Garey and Johnson, 2003], both for the un-weighted case and for the weighted one.

## 2.2 Modeling a tree

Let T be a binary tree, E the set of its external nodes, I the set of its internal nodes, V the union of E and I ($V = E \cup I$), B the set of its branches. Let's assume $\mid E \mid = n$ and $\mid I \mid = m$.[5]

We can represent T using a $|I| \times |E \cup I|$ matrix M, whose elements are defined in this way:

$$m_{ij} = \begin{cases} 1, & \text{if node } i \text{ is father of node } j; \\ 0, & \text{otherwise.} \end{cases} \tag{2.12}$$

This kind of representation begins to be inefficient when the number of columns of the matrix $M$ becomes high. In fact, in this case, since the maximum number of elements $m_{ij}$ equal to one is at most two[6] for each $i$, the number of elements $m_{ij}$ of $M$ equal to zero is enormous in proportion to the unitary elements.

To improve the efficiency we can: compress the matrix $M$ reducing the number of columns to two, label each element of $I \cup E$ using a positive integer number $x$ (label) between zero and $2n-1$, and consider a new matrix $M^{'}$ of dimension $(n-1) \times 2$ and such that

$$m^{'}_{ij} = label(i,j) \quad i = 0...n-1, j = 0...1 \tag{2.13}$$

where $label(i,j)$ is a function that, varying $j$, returns the label $x$ of the left or the right child of the internal node $i$. A graphical example of this kind of representation is given in Fig. 2.2.

[4]It must be noted that we don't want only to find the topology of the tree, but also the labels of the leaves and the sequences of the internal nodes.

[5]We remind to the reader that—as we saw in the first chapter—the number of internal nodes is $m = n - 1$.

[6]This because the tree is supposed binary.

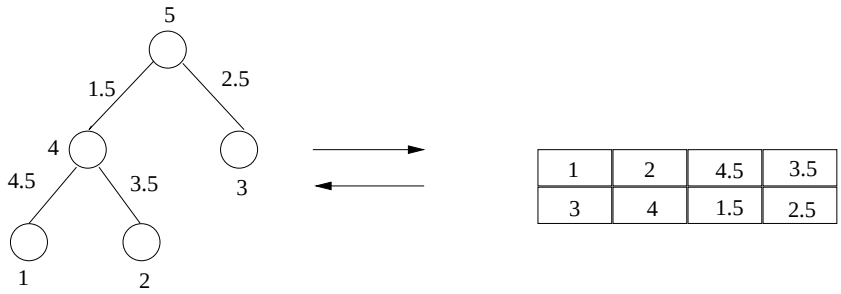


Figure 2.2: Representation of a tree in form of rectangular matrix.

## 2.3 Neighborhood definition

A critical point during the modeling of the problem is the definition of the neighborhood. This aspect influences directly the computing time necessary to describe the solutions space, and indirectly the quality of the solutions found. A good definition of neighborhood is often a trade-off between these two factors. To understand the combinatorial nature of this problem is fundamental for modeling neighborhood operators (good references are [Dress, 2001, 2000, Roberts and Sheng, 2000, Ahuja et al., 2001, Scaparra et al., 2003, Pe'er et al., 2004]). We have used these references for defining new concepts of neighborhood of a solution.

In the following, we will assume that all external nodes can be labeled by a positive integer number in [0, n-1][7] and respectively for the internal nodes in [n, 2n-1].

We give the following definitions of neighborhood:

**Definition 2.1** *Let $S$ be the solutions space. We define the constant internal topology neighborhood (CITN) of a matrix $M$, the set $S_M^E = \{M' \in S : m'_{ij} = m_{ij} \ \forall \ i,j : label(i,j) > n-1\}$.*

**Definition 2.2** *Let $S$ be the solutions space. We define the variable internal topology neighborhood (VITN) of a matrix $M$, the set $S_M^I = \{M'' \in S : m''_{ij} = m_{ij} \ \forall \ i,j : label(i,j) \leq n-1\}$.*

**Definition 2.3** *Let $S$ be the solutions space. We define the totally constructive topology neighborhood (TCTN) of a matrix $M$, the set $S_M^T = \{M''' \in S : m'''_{ij} \neq m_{ij}$ at least for a couple $(i,j)\}$.*

We define also the following operators:

[7] I.e., given the set of external nodes $\{e_0, e_1, ..., e_{n-1}\}$, we can represent this set as a sequence {0,1,...,n-1}.

**Definition 2.4** *We define operator* $\nabla^E : S \rightarrow S^E_M$ *each operation that, modifying a solution* $M \in S$*, returns a solution* $M^{'} \in S^E_M$*.*

**Definition 2.5** *We define operator* $\nabla^I : S \rightarrow S^I_M$ *each operation that, modifying a solution* $M \in S$*, returns a solution* $M^{''} \in S^I_M$*.*

**Definition 2.6** *We define operator* $\nabla^T : S \rightarrow S^T_M$ *each operation that, modifying a matrix* $M \in S$*, returns a matrix* $M^{'''} \in S^T_M$*.*

**Definition 2.7** *We define permutations of n distinct elements, all the various groupings that can be formed with the given elements, respecting the following property:*

1. *Each group contains* n *elements.*
2. *Each element can appear in the group once and only once time.*
3. *Two groups differ only for the order in which the elements appear.*

**Definition 2.8** *Given n distinct elements and an integer k,* $k \in \mathbb{Z}^+$*,* $k \leq n$*, we call simple combinations of n elements of class k, all the various groupings that can be formed with the given elements, so that the following property are valid:*

1. *Each group contains k elements.*
2. *Each element can appear in the group once and only once time.*
3. *Two groups that differ at least for an element are considered differents. The order is not important.*

### 2.3.1 Models $\nabla^E$

This operator does not modify the internal topology of the tree but only the configuration of the leaves. In this way, $\nabla^E$ allows the generation of solutions very similar among them, and therefore the local improvement of the quality of the solutions found. In the following we give two pseudo codes describing two examples of $\nabla^E$ operators: $\nabla^E_\mu$ and $\nabla^E_{macro}$. They differ only in the way they manage the leaves: $\nabla^E_\mu$ works using combinations of leaves, $\nabla^E_{macro}$ works using permutations of leaves. In Figure 2.3 and 2.4 we report their pseudocodes.

**procedure** $\nabla^E_\mu$
$k \leftarrow$SelectIntegerRandomIn$(2, n-1)$
$L \leftarrow$Select and remove $k$ elements in the current solution $M$ such that $label(i,j) \leq n-1$ and set the respective $M_{ij} = -1$
**for** each $M_{ij} = -1$ **do**
  L$\leftarrow$Permutation(L)
  Select a random element $q$ in L
  $M_{ij} \leftarrow L_q$
  Remove $L_q$
**end for**

Figure 2.3: Modifying the assignment of a subset of leaves to internal nodes, $\nabla^E_\mu$ allows the generation of solutions very similar among them, and therefore the local improvement of the quality of the solutions found.

**procedure** $\nabla^E_{macro}$
$L \leftarrow$Select and remove all elements in the current solution $M$ such that $label(i,j) \leq n-1$ and set the respective $M_{ij} = -1$
**for** each $M_{ij} = -1$ **do**
  L$\leftarrow$Permutation(L)
  Select a random element $q$ in L
  $M_{ij} \leftarrow L_q$
  Remove $L_q$
**end for**

Figure 2.4: $\nabla^E_{macro}$ generates new solutions modifying the assignment of all the leaves. Comparing $\nabla^E_\mu$ and $\nabla^E_{macro}$ we can observe as the latter allows the generation of solutions potentially more different among them then the former. At the contrary, the former allows the generation of solutions more similar among them. Therefore we can use the former if the goal is to improve locally an assignment of a subset of leaves, and the latter if the goal is to obtain solutions characterized by different assignments of leaves.

### 2.3.2 Model $\nabla^I$

This operator does not modify the external assignment of the leaves of the tree but only its internal topology. In this way, $\nabla^I$ allows the local improvement of the quality of the internal topological structure relative to a solution. In Figure 2.5 we report the pseudocode of $\nabla^I$ operator.

### 2.3.3 Model $\nabla^T$

This operator modify both of external assignment of the leaves and the internal topological structure of the tree. $\nabla^T$ is a destructive operator, because it does not maintain any information about the topological structure of the tree. $\nabla^T$ can be used in order to direct the search in new different area of the solutions space $S$.In Figure 2.6 we report the pseudocode of $\nabla^T$ operator.

## 2.4 Evaluating a solution

Another critical point for the phylogenetic inference problem is the definition of a quality function. To define it is not a simple problem because it is difficult to give a concept of "goodness of a solution".

There exist different quality functions in the literature; in this works we have used the *minimum total cost of a tree* (formula 2.11). This function is normally used as quality function in the weighted small parsimony problem (WSPP) [Swofford et al., 1996, Shamir and Naor, 2002], and the Sankoff's algorithm shows how compute it. We describe it in the following.

Let's assume to have a set $\Gamma$ of $n$ leaves representing species, a set of $m$ characters describing the DNA of each species, a set of $k$ states for each character, and a matrix[8] $C^c_{ij}$ representing the cost for changing from state $i$ to state $j$ relative to the character $c$.

**Definition 2.9** *We define cost of a leaf $v$, for each character $c$ and state $t$, the number*

$$S^c_t(v) = \begin{cases} 0, & v_c = t, \\ \infty, & otherwise. \end{cases} \tag{2.14}$$

**Definition 2.10** *We define cost of an internal node $w$, with children $u$ and $z$, for each character $c$ and state $t$, the number*

$$S^c_t(w) = min_i\{C^c_{ti} + S^c_i(u)\} + min_j\{C^c_{tj} + S^c_j(z)\} \tag{2.15}$$

[8]In the literature there exist many models for this matrix. The reader can find them in Swofford et al. [1996].

**procedure** $\nabla^I$
$L \leftarrow$ Remove all elements of the rows in which
there exists at least one element such that
$label(i, j) > n - 1$ and set the respective $M_{ij} = -1$
**for** each $M_{ij} = -1$ **do**
  L$\leftarrow$Permutation(L)
  Select a random element $q$ in L such that $L_q < i$
  $M_{ij} \leftarrow L_q$
  Remove $L_q$
**end for**

Figure 2.5: Pseudocode relative to the $\nabla^I$ operator. It allows the local improvement of the quality of the internal topological structure relative to a solution.

**procedure** $\nabla^T$
$L \leftarrow$ Remove all elements of the solution $M$ and insert
in L only the elements such that $label(i, j) \leq n - 1$
**for** each row $i$ of the solution $M$ **do**
  $L \leftarrow$Permutation(L)
  select two nodes in L
  add node $i$ to L
**end for**

Figure 2.6: Pseudocode relative to the $\nabla^T$ operator. It can be used in order to direct the search in new different area of the solutions space $S$.

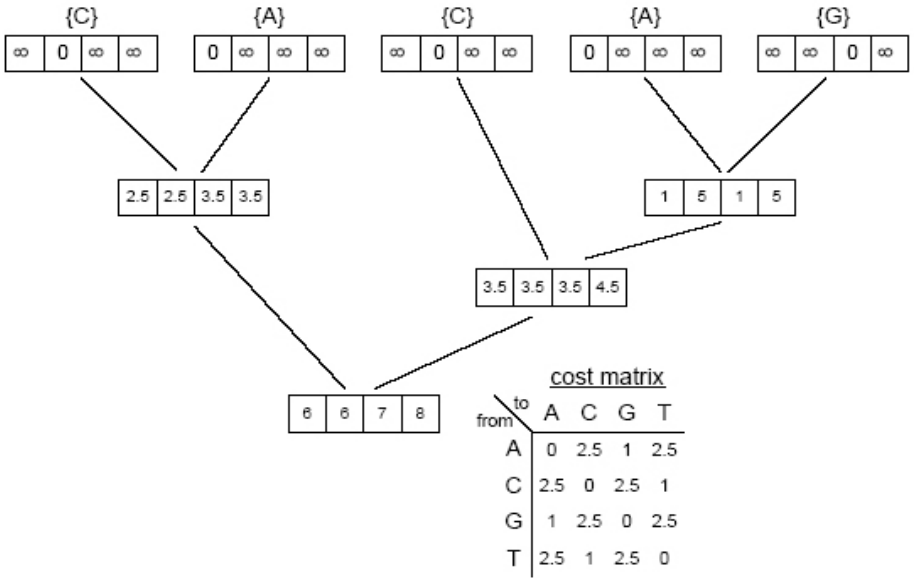


Figure 2.7: An example illustrating the computation of the cost of a tree.

**Definition 2.11** *We define minimum total cost of a tree $T$ with root node $r$ the number*

$$S(T) = \sum_{c=1}^{m} min_t S_t^c(r) \tag{2.16}$$

**Sankoff's algorithm complexity:** Repeating the formula 2.14 for each state of each character of each leaf, we do $n \cdot k \cdot m$ steps; repeating the formula 2.15 we do $8 \cdot m \cdot k \cdot (n-1)$ steps; the resulting computational cost is $O(9 \cdot m \cdot k \cdot n) \simeq O(\alpha \cdot n \cdot m)$, where $\alpha = 36$ if we assume that the number of all the possible states is four, $\alpha = 135$ if we consider that the number of all possible states is fifteen (see the table in Figure 2.8).

We can improve the computational cost of the Sankoff's algorithm by observing that the result of the application of the formula 2.14, for each character of each leaf, is independent of the internal topological structure of the evaluating tree. Defining therefore the vectorial space on the real numbers

$$\mathbb{M}_{n\times m}^{(\mathbb{R})} = \{(a_{ij}) \in \mathbb{R} : \exists a_{ij} \to \infty, \quad i = 1...n \quad j = 1...m\} \tag{2.17}$$

and considering the matrix $\mathbf{H} \in \mathbb{M}_{n\times m}^{(\mathbb{R}^4)}$, such that the generic element $h_{ij}$ is equal to the vector obtained by the table in Figure 2.8 in correspondence

| state | glossary | vector |
|---|---|---|
| A | A | $\{0, \infty, \infty, \infty\}$ |
| T | T | $\{\infty, 0, \infty, \infty\}$ |
| C | C | $\{\infty, \infty, 0, \infty\}$ |
| G | G | $\{\infty, \infty, \infty, 0\}$ |
| M | $A \vee C$ | $\{0, \infty, 0, \infty\}$ |
| R | $A \vee G$ | $\{0, \infty, \infty, 0\}$ |
| W | $A \vee T$ | $\{0, 0, \infty, \infty\}$ |
| S | $C \vee G$ | $\{\infty, \infty, 0, 0\}$ |
| Y | $C \vee T$ | $\{\infty, 0, 0, \infty\}$ |
| K | $G \vee T$ | $\{\infty, 0, \infty, 0\}$ |
| V | $A \vee C \vee G$ | $\{0, \infty, 0, 0, \}$ |
| H | $A \vee C \vee T$ | $\{0, 0, 0, \infty\}$ |
| D | $A \vee G \vee T$ | $\{0, 0, \infty, 0\}$ |
| B | $C \vee G \vee T$ | $\{\infty, 0, 0, 0\}$ |
| N | $A \vee C \vee G \vee T$ | $\{0, 0, 0, 0\}$ |

Figure 2.8: This table represents the way in which it is possible representing the standard ambiguity code using a vector of four components.

to the $j$-th character of the DNA sequence of the $i$-th species; defining: the operator $\oplus : \mathbb{M}^{(\mathbb{R})}_{\theta_1 \times \theta_2} \to \mathbb{M}^{(\mathbb{R})}_{\theta_1 \times \theta_2}$, $\theta_1 \wedge \theta_2 \in \mathbb{N}$, as follows:

$$\forall \mathbf{V} \in \mathbb{M}^{(\mathbb{R})}_{\theta_1 \times \theta_2}, \mathbf{C} \in \mathbb{M}^{(\mathbb{R})}_{\theta_1 \times \theta_2} \quad \mathbf{V} \oplus \mathbf{C} = \mathbf{L} \in \mathbb{M}^{(\mathbb{R})}_{\theta_1 \times \theta_2} \tag{2.18}$$

$$l_{ij} = v_{ij} + c_{ij}, \quad i = 1...\theta_1, j = 1...\theta_2 \tag{2.19}$$

the operator $\circlearrowleft$: $\mathbb{R}^\nu \to \mathbb{M}^{(\mathbb{R})}_{\nu \times \nu}$, $\nu \in \mathbb{N}$, as follows:

$$\forall \vec{v} \in \mathbb{R}^\nu, x^{\circlearrowleft} = \mathbf{\Lambda} \in \mathbb{M}^{(\mathbb{R})}_{\nu \times \nu}, \quad \lambda_{ij} = \lambda_{kj} \quad j = 1..\nu, i \wedge k = 1...\nu, i \neq k \tag{2.20}$$

the operator $\| \cdot \|_{min} : \mathbb{M}^{(\mathbb{R})}_{\phi \times \phi} \to \mathbb{R}^\phi$, $\phi \in \mathbb{N}$ as follows:

$$\forall \mathbf{P} \in \mathbb{M}^{(\mathbb{R})}_{\phi \times \phi} \qquad \|\mathbf{P}\|_{min} = \vec{q}, \qquad \vec{q} \in \mathbb{R}^\phi, \tag{2.21}$$

$$q_i = min_j P_{ij} \quad i = 1...\phi, j = 1...\phi \tag{2.22}$$

the matrix

$$\mathbf{\Omega} = \begin{pmatrix} \vec{\omega}_1 \\ \vec{\omega}_2 \\ ... \\ \vec{\omega}_n \end{pmatrix}, \vec{\omega}_i = \begin{pmatrix} \omega_{i1} \\ \omega_{i2} \\ ... \\ \omega_{im} \end{pmatrix}^T \quad \omega_{ij} \in \mathbb{R}^4 : \omega_{ij} = \|h^{\circlearrowleft}_{ij} \oplus \mathbf{C}^j\| \tag{2.23}$$

**procedure** Select
**input**: $i$ =integer, **output**: $\vec{v} \in \mathbb{R}^{4 \cdot m}$
**if** $i < n$ **then**
  **return** $\vec{\omega}_i$
**else**
  **return** $\vec{\psi}_{i-n}$
**end if**

**procedure** Evaluate
**input**: a solution $M$, **output**: $r \in \mathbb{R}$
**for** $i = 1$ to $n - 1$ **do**
  $\vec{v}_1 \leftarrow \text{Select}(m_{i1})$
  $\vec{v}_2 \leftarrow \text{Select}(m_{i2})$
  $\vec{\psi}_i = \vec{v_1} \oplus \vec{v_2}$
  **for** $j = 1$ to $m$ **do**
    $\vec{\psi}_{ij} = \|\vec{\psi}_{ij}^{\circlearrowleft} \oplus \mathbf{C}^j\|_{min}$
  **end for**
**end for**
**return** $\sum_{j=1}^{m} min_{k=1..4} \psi_{ijk}$

Figure 2.9: This is the pseudocode relative to the algorithm that computes the quality function of a solution.

where $\mathbf{C}^j$ is the cost matrix relative to the character $j$ of the DNA sequence, and the matrix

$$\mathbf{\Psi} = \begin{pmatrix} \vec{\psi}_1 \\ \vec{\psi}_2 \\ \dots \\ \vec{\psi}_{n-1} \end{pmatrix}, \vec{\psi}_i = \begin{pmatrix} \psi_{i1} \\ \psi_{i2} \\ \dots \\ \psi_{im} \end{pmatrix}^T, \quad \psi_{ij} \in \mathbb{R}^4 \tag{2.24}$$

we can compute the quality function using the algorithm described in Figure 2.9.

This new algorithm needs of $O((n-1) \times (4 \cdot m + 16 \cdot m)) \simeq O(\alpha \cdot m \cdot n)$ steps, with $\alpha$=20. So pre-computing the matrix $\mathbf{\Omega}$ allows us almost to halve, in the worst case, the constant $\alpha$ of the Sankoff's algorithm.

# Chapter 3

# Solving techniques

In this chapter we describe the solution techniques we have used for solving the considered problem. We have split each section of this chapter in two parts: the first part is dedicated to the description of the methodology used, the second part describes the details of each implementation. If the reader needs more details about the methodologies used, he can find them in [Glover and Kochenberger, 2003, Dorigo and Stützle, 2004].

## 3.1 Simulated annealing

Simulated annealing (SA) is a generalization of a Monte Carlo method for examining the equations of state and frozen states of $n$-body systems [Metropolis et al., 1953]. The concept is based on the manner in which liquids freeze or metals recrystalize in the process of annealing. In an annealing process a melt, initially at high temperature, is slowly cooled so that the system at any time is approximately in thermodynamic equilibrium. As cooling proceeds, the system becomes more ordered and approaches a "frozen" ground state at $T = 0$. Hence the process can be thought of as an adiabatic approach to the lowest energy state. If the initial temperature of the system is too low or cooling is done insufficiently slowly the system may become quenched forming defects or freezing out in metastable states (i.e., trapped in a local minimum energy state). In the original Metropolis schema we select an initial state of a thermodynamic system, characterized by the internal energy E and temperature T. Holding T constant, we perturb the initial configuration and compute the change in energy dE. If the change in energy is negative, the new configuration is accepted, otherwise, if the change in energy is positive, it is accepted with a probability given by the Boltzmann factor exp -(dE/T). This processes is repeated a sufficient number of times to give good

sampling statistics for the current temperature, and then the temperature is decremented and the entire process repeated until a frozen state is achieved at T=0.

The generalization of this Monte Carlo approach to combinatorial problems is straightforward [Kirkpatrick et al., 1983]. The current state of the thermodynamic system is analogous to the current solution to the combinatorial problem, the energy equation for the thermodynamic system is analogous to at the objective function, and ground state is analogous to the global minimum. The major difficulty in the implementation of the algorithm is that there is no obvious analogy for the temperature T with respect to a free parameter in the combinatorial problem. Furthermore, avoidance of entrainment in local minima is dependent on the "annealing schedule", the choice of initial temperature, how many iterations are performed at each temperature, and how much the temperature is decremented at each step as cooling proceeds.

Figure 3.1 gives a general algorithmic outline for SA. We have indicated with $s$ the initial solution, with $s'$ the neighbor solution of $s$ (often it is generated randomly according to a uniform distribution), with $T$ the temperature, and with $f(\cdot)$ the objective function. To implement a simulated annealing algorithm, the following parameters and functions have to be specified:

- The function `GenerateInitialSolution`, that generates an initial solution.
- The function `InitializeAnnealingParameters` that initializes several parameters used in the *annealing schedule*; the parameters comprise
  - an initial temperature $T_0$,
  - the number of iterations to be performed at each temperature (inner loop criterion in Figure 3.1),
  - a termination condition (outer loop criterion in Figure 3.1).
- The function `GenerateNeighbor` that chooses a new solution $s'$ in the neighborhood of the current solution $s$.
- The function `AcceptSolution` that implements the following equation

$$p_{accept}(s, s', T) = \begin{cases} 1, & \text{if } f(s') < f(s); \\ exp(\frac{f(s)-f(s')}{T}), & \text{otherwise.} \end{cases} \quad (3.1)$$

  that describes the probability of accepting a neighbor solution $s'$.
- The function `UpdateTemp` that returns a new value for the temperature.

```
procedureSimulatedAnnealing
s ← GenerateInitialSolution
InitializeAnnealingParameters
s_best ← s
n ← 0
while (outer-loop termination condition not met) do
  while (inner-loop termination condition not met) do
    s' ← GenerateNeighbor(s)
    s ←AcceptSolution(T_n,s,s')
    if (f(s) < f(s_best)) then
      s_best ← s
    end if
  end while
  UpdateTemp(n); n ← n + 1
end while
```

Figure 3.1: High-level pseudo-code for simulated annealing (SA).

### 3.1.1 SA algorithm for inferring phylogenies

Our implementation of the SA algorithm to tackle the phylogenetic inference problem has the following features:

- The function `GenerateInitialSolution` is external to the SA algorithm and in common with the other metaheuristics; an outline pseudo-code for the generation of the initial solution is given in Chapter 5.
- The function `InitializeAnnealingParameters` that initialize the parameters used in the annealing schedule is characterized by:
  - the initial temperature $T_0$ set to 100,
  - the number $L$ of iteration to be performed at each temperature (inner loop criterion in Figure 3.1) is function[1] of the initial temperature $T_0$, and of the current temperature $T$:

$$L = \begin{cases} \texttt{Random}\cdot T_0, & T - T_0 = 0; \\ \texttt{Map}(T - T_0)\cdot\texttt{Random}\cdot T_0, & T - T_0 \neq 0. \end{cases} \tag{3.2}$$

  - the termination condition (outer loop criterion in Figure 3.1) is based on temporal criterion: if $t_{in}$ is the initial time, $t$ is the current time, and $t_{max}$ denotes the maximum time allowed for computation, the termination condition can be written in the following

[1]The function `Random` of the equation 3.2 returns a random number in [0,1]; the function `Map` maps a number $x \in \mathbb{R}$ in [0,1].

way:

`bool TerminationCondition(){ return (` $t > t_{max} - t_{in}$ `)}`

- The function `GenerateNeighbor` works in this way: initially the operator used for generating the neighborhood is $\nabla^E_\mu$ and the constant $a$ (used for updating the temperature $T$, see function `UpdateTemp`) is set to 0.99. This configuration is used during the temporal interval $[0, \frac{k}{4}]$, where $k \in \mathbb{Z}$ is a constant given by the user such that $k < t_{max}$. If the cost function does not change in $[0, \frac{k}{4}]$, during the temporal interval $[\frac{k}{4}, \frac{k}{2}]$ we set $a = 1.1$ and use $\nabla^E_{macro}$ for generating the neighborhood. If no variation is noted in $[\frac{k}{4}, \frac{k}{2}]$, during the temporal interval $[\frac{k}{2}, \frac{3k}{2}]$ we proceed using the operator $\nabla^I$ and setting $a = 0.99$, that is we proceed trying to find the local optimal internal topology. If the cost function does not change in $[\frac{k}{2}, \frac{3k}{4}]$, during the interval $[\frac{3k}{4}, k]$ we set $a = 1.1$ and use $\nabla^T$ operator for generating the neighborhood. If we have at least a modification in the cost function in whatever temporal interval $t' \in [0, k]$, this loop restart again, setting $a = 0.99$ and using $\nabla^E_\mu$ as neighborhood generator.

- The function `AcceptSolution` is the same of the equation 3.1.

- The function `UpdateTemp` update the temperature $T$ at time $t$ in the following way: $T = a^t \cdot T$, where $a = \begin{cases} 0.99, & \text{if the operator used} \\ & \text{is } \nabla^E_\mu \text{ or } \nabla^I; \\ 1.1, & \text{otherwise.} \end{cases}$

## 3.2 Tabu search

The basic concept of tabu search (TS) as described by [Glover, 1986, Glover and Laguna, 1997] is a meta-heuristic superimposed to an heuristic. The main idea is to avoid cycling by forbidding or penalizing moves which take the solution, in the next iteration, to points in the solution space previously visited—hence "tabu". The tabu search is quite old, Glover attributes it's origin to about 1977. The method is still actively researched, and is continuing to evolve and improve. The tabu search method was partly motivated by the observation that human behavior appears to operate with a random element that leads to inconsistent behavior given similar circumstances [Glover and Laguna, 1997]. As Glover points out, the resulting tendency to deviate from a charted course, might be regretted as a source of error but can also prove to be source of gain. The tabu method operates in this way with the exception that new courses are not chosen randomly. Instead the tabu search

proceeds according to the hypothesis that there is no point in accepting a new (poor) solution unless it is to avoid a path already investigated. This insures new regions of a problems solution space will be investigated, with the goal of avoiding local minima and ultimately finding the desired solution. The tabu search begins by moving to a local minimum. To avoid cycling, the method records recent moves in one or more tabu lists. The original intent of the list was not to prevent a previous move from being repeated, but rather to insure it was not reversed. The tabu lists are historical in nature and form the tabu search memory. The role of the memory can change as the algorithm proceeds. At initialization the goal is to make a coarse examination of the solution space, but as candidate locations are identified the search is more focused to produce local optimal solutions in a process of 'intensification'. In many cases the differences between the various implementations of the tabu method have to do with the size, variability, and adaptability of the tabu memory to a particular problem domain.

Figure 3.2 gives a general algorithmic outline for a simple tabu search. We have indicated with $s$ the initial solution, and with $f(\cdot)$ the objective function. The functions needed to define it are the following:

- The function `GenerateInitialSolution`, that generates an initial solution.
- The function `InitializeMemoryStructures`, that initializes all the memory structures used during the run of the TS algorithm.
- The function `GenerateAdmissibleSolutions`, that is used to determine the subset of neighbor solutions which are not tabu.
- The function `SelectBestSolution`, that returns the best admissible move.
- The function `UpdateMemoryStructures`, that updates the memory structures.

### 3.2.1 TS algorithm for inferring phylogenies

Our implementation of the TS algorithm to tackle the phylogenetic inference problem has the following features:

- The function `GenerateInitialSolution` is external to the TS algorithm and in common with the other metaheuristics; an outline pseudocode for the generation of the initial solution is given in Chapter 5.

```
procedureTabuSearch
s ← GenerateInitialSolution
InitializeMemoryStructures
s_best ← s
while ( termination condition not met) do
  A ← GenerateAdmissibleSolutions(s)
  s ←SelectBestSolution(A)
  UpdateMemoryStructures
  if (f(s) < f(s_best)) then
    s_best ← s
  end if
end while
```

Figure 3.2: High-level pseudo-code for tabu search (TS).

- The function `InitializeMemoryStructures` creates a tabu list with an initial length of $l = \lceil\sqrt{|E|}\rceil$, as is usual in many implementation of the tabu search. In the tabu list we don't store movements but solutions, and we define tabu all those solutions that are already in the tabu list.

- The function `GenerateAdmissibleSolutions` works in the following way: let $max$ be a positive integer given by the user, $\alpha, \beta, \gamma, \delta$ positive integer such that $\alpha+\beta+\gamma+\delta = max$. The neighborhood of a solution $s$ is computed by applying $\alpha$ times the operator $\nabla^E_\mu$, $\beta$ times the operator $\nabla^E_{macro}$, $\gamma$ times the operator $\nabla^I$ and $\gamma$ times the operator $\nabla^T$ to the solution $s$. We use the following table for setting the parameters $\alpha, \beta, \gamma, \delta$:

| | $\alpha$ | $\beta$ | $\gamma$ | $\delta$ |
|---|---|---|---|---|
| $[0, \frac{k}{4}]$ | $\frac{MAX}{4}$ | $\frac{MAX}{4}$ | $\frac{MAX}{4}$ | $\frac{MAX}{4}$ |
| $[\frac{k}{4}, \frac{k}{2}]$ | 0 | $\frac{MAX}{4}$ | $\frac{MAX}{4}$ | $\frac{MAX}{2}$ |
| $[\frac{k}{2}, \frac{3k}{4}]$ | 0 | $\frac{MAX}{2}$ | 0 | $\frac{MAX}{3}$ |
| $[\frac{3k}{4}, k]$ | 0 | 0 | 0 | $MAX$ |

(3.3)

  where $k$ is the same of the SA implementation. The values assigned to $\alpha$, $\beta$, $\gamma$, $\delta$ in the table don't derive by a particular study of tuning of these value, but have been selected in according to thumb rules. These arguments anyway will be object of the next study.

- The function `SelectBestSolution` is implemented as a simple loop that returns the position and the value of the solution in the neighbor-

hood with the smallest value. The following is the outline pseudo-code for the function:

**procedure** SelectBestSolution
min=$\infty$, val=$\infty$
**for** each solution $s$ in the neighborhood **do**
  **if** $f(s) < min$ **then**
    min=s; val=$f(s)$;
  **end if**
**end for**
**return** min and val

- The function `UpdateMemoryStructures` add the non-tabu solution in the tabu list, and modifies the tabu list length varying its dimension between $[0.25l, 2l]$, in function of the value of the $f_{best}$ (the value of the best-so-far solution): if $f_{best}$ remains constat during the interval $[t', t'']$ such that $(t'' - t') > const$, where *const* is a parameter given by the user, a random number $r$ between [0.25, 2] is selected and the tabu list length is set to $r$.

## 3.3 Iterated local search

Iterated local search (ILS) [Lourenço et al., 2002] is a simple metaheuristic working as follows. Starting from an initial solution S, a local search is applied. Once the local search is stuck, the locally optimal solution $\hat{s}$ is perturbed by a move in a neighborhood different from the one used by the local search. This perturbed solution $s'$ is the new starting solution for the local search that takes it to the new local optimum $\hat{s}'$. Finally, an acceptance criterion decides which of the two locally optimal solution to select as a starting point for the next perturbation step. The main motivation for ILS is to build a randomized walk in a search space of the local optima with respect to some local search algorithm.

An algorithmic outline of ILS is given in Figure 3.3. The four functions needed to specify an ILS algorithm are:

- The function `GenerateInitialSolution` that generates an initial solution.
- The function `LocalSearch` that returns a locally optimal solution $\hat{s}$ when applied to $s$.
- The function `Perturbation` that perturbs the current solution $s$ generating an intermediate solution $s'$.

```
procedureIteratedLocalSearch
s ← GenerateInitialSolution
ŝ ←LocalSearch(s)
s_best ← ŝ
while ( termination condition not met) do
  s' ← Perturbation(ŝ)
  ŝ' ←LocalSearch(s')
  if (f(s) < f(s_best)) then
    s_best ← ŝ'
  end if
  ŝ ← AcceptanceCriterion(ŝ, ŝ')
end while
```

Figure 3.3: High-level pseudo-code for iterated local search (ILS).

- The function `AcceptanceCriterion` that decides from which solution the search is continued at the next perturbation step.

### 3.3.1 ILS algorithm for inferring phylogenies

Our implementation of the ILS algorithm to tackle the phylogenetic inference problem has the following features:

- The function `GenerateInitialSolution` is external to the ILS algorithm and in common with the other metaheuristics; an outline pseudo-code for the generation of the initial solution is given in Chapter 5.

- The function `LocalSearch`, that returns a locally optimal solution, works in this way: let $s$ be the current solution, $max$ a positive integer given by the user, $\alpha$,$\beta$,$\gamma$,$\delta$ positive integers such that $\alpha + \beta + \gamma + \delta = max$; the solution $\hat{s}$ is computed by applying $\alpha$ times the neighborhood operator $\nabla^E_\mu$, $\beta$ times the neighborhood operator $\nabla^E_{macro}$, $\gamma$ times the neighborhood operator $\nabla^I$, $\delta$ times the neighborhood operator $\nabla^T$. The parameters $\alpha$,$\beta$,$\gamma$,$\delta$, are selected using the table 3.3.

- The function `Perturbation` is the most delicate part of the ILS algorithm. In fact, "if the perturbation is too strong, ILS may behave like a random restart, so better solution will only be found with a very low probability. On the other hand, if the perturbation is too small, the local search will often fall back into the local optimum just visited and the diversification of the search space will be very limited"[Lourenço et al., 2002]. We implemented this function using only two operators:

$\nabla^E_\mu$ and $\nabla^I$: given the solution $s$, we apply before the former operator, and after the latter; in other words we try to perturb the solution with respect both the external nodes and the internal nodes.

- The function `AcceptanceCriterion` decides in the following way from which the search is continued at the next perturbation step: if $f(\hat{s}) < f(\hat{s}')$, the function returns $\hat{s}$, $\hat{s}'$ otherwise.

## 3.4 Ant colony optimization

Ant colony optimization (ACO) [Dorigo and Stützle, 2004] is a metaheuristic in which a colony of artificial ants cooperate in finding good solutions to discrete optimization problems. Each ant, represented as a simple agent, disposes of computational resources, and communicates indirectly with the other ants by stigmergy, that is, by indirect communication (through the pheromone) mediated by the environment. The building process, that is the process by which each ant builds solutions by moving on the space of the solutions' components, is stochastic and exploits both pheromone trails and heuristic values for making probabilistic decisions on how to move on the space of the solutions' components. The pheromone trails encode a long-term memory about the entire ants search processes, and are updated by the ants themselves. The heuristic values represent, on the contrary, a priori information about the problem instance or run time information provided by a source different from the ants.

To implement ACO is necessary:

- To define the information represented by the pheromone and the data structures representing it.
- To define the stochastic procedure for making decision on how to move on the space of the solutions' components, in particular:
  - to define a model embodying the information about the pheromone trails and the heuristic values,
  - to define a strategy for selecting a solution component.
- To define the procedure for managing the pheromone.

ACO, of which a pseudo-code is given in Figure 3.4, is made up of three procedures: `ConstructAntSolutions`, `UpdatePheromones`, `DaemonActions`.

`ConstructAntSolutions` represents the process by which a colony of ants builds concurrently and asynchronously new solutions for the considered problem. The building process, simulating the movement of the ants,

is based on a stochastic local decision policy that makes use of pheromone trails and heuristic information.

`UpdatePheromones` represents the process by which the pheromone trails are modified. The trail values can either increase, as ants deposit pheromone on the components and/or connections they use, or decrease, due to pheromone evaporation.

Finally, `DaemonActions` is used to implement centralized actions which cannot be performed by a single ants. Example of daemon actions are the activation of a local optimization procedure, or collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search from a non-local perspective (see [Dorigo and Stützle, 2004] for more information).

**procedure**AntColonyOptimization
**while** (termination condition not met) **do**
  `ConstructAntSolutions`
  `UpdatePheromones`
  `DaemonActions`
**end while**

Figure 3.4: High-level pseudo-code for ant colony optimization (ACO).

### 3.4.1 ACO algorithm for inferring phylogenies

Our implementation of the ACO algorithm to tackle the phylogenetic inference problem is the following:

- Let $I$ be the set of internal nodes, $E$ the set of external nodes, and $V = E \cup I$ the set of all nodes. Let's assume that for each external node $e \in E$ there exists an edge for each internal node $i$ (see Figure 3.5). Let's consider now the connected, un-weighted, and un-direct graph $G(V, A)$ where $A$ is the set of edges; since the graph is un-weighted no heuristic information can be used, therefore only the pheromone trails lead the colony of ants during the building process. We have modeled the pheromone trails as a matrix $\Theta \in \mathbb{M}^{\mathbb{R}}_{|I|\times|E\cup I|}$ whose elements $\tau_{ij} \in [0, 1]$.

- The stochastic procedure for making decision on how to move on the space of the solutions ' components depends by the type of neighborhood selected: CITN, VITN, TCTN.

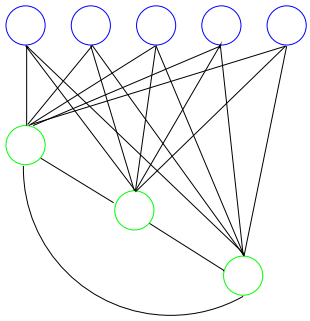

Figure 3.5: Graph relative to five taxa; the blue circles represent the external nodes, the green circles represents the internal nodes. The graph is un-weight, un-direct, and connected.

**CITN:** in this case $k \leq |E|$ external nodes are selected[2], removed by the tree, and inserted in a list $L_{eCITN}$; all the corresponding internal nodes $i$, to which the $j$-th removed external nodes was linked, are inserted in the list $L_{iCITN}$. By introducing $p_{ij}$ as the probability that the internal node $i$ links the node $j$, we can consider, for each node $i \in L_{iCITN}$, the vector $\vec{p}_i$ whose components, for each $j \in L_{eCITN}$, are computed in this way:

$$p_{ij} = \frac{\tau_{ij}}{\sum_{l \in L_{eCITN}} \tau_{il}}. \tag{3.4}$$

The sum of the components $p_{ij}$ of the vector $\vec{p}_i$ is equal to one so, if $r$ is a random number uniformly distributed in [0,1], we can select the $j$-th and the $(j+1)$-th component of $\vec{p}_i$ using the following pseudo-code[3]:

Let b=0;j=0;
**while** $b < r$ or $L_{eCITN_j} \geq i + n - 1$ **do**
  $b \leftarrow p_{ij}$; $j \leftarrow j + 1$
**end while**

It must be noted that if $k < |E|$ this procedure is the analogous of the operator $\nabla^E_\mu$, while if $k = |E|$ this procedure is the analogous of the operator $\nabla^E_{macro}$.

[2]We consider therefore assigned the remaining $|I| + |E| - k$ nodes.

[3]This pseudo-code must be repeated for the $j$-th and for the (j+1)-th component because we we want to select two children for each internal node at each step of the building process.

**VITN:** the stochastic procedure is similar to CITN with the only difference that are removed all the internal nodes having at least a child that is an internal node. The internal nodes having two external nodes as children are not removed. It must be noted that this kind of building process works in the same way of the $\nabla^I$ operator.

**TCTN:** let $L$ be the list of the nodes (both internal and external) of the tree, and $\Lambda$ a list of boolean with length equal to $L$. If the $j$-th entry of $\Lambda$ equals to *true* then we will consider the $j$-th node of $L$ as part of the solution we are building. By introducing $p_{ij}$ as the probability that the internal node $i$ links the node $j$, we can consider, for each internal node $i \in I$, the vector $\vec{p}_i$ whose components, for each external node $j \in E$, are computed in this way:

$$p_{ij} = \begin{cases} \frac{\tau_{ij}}{\sum_{l \in \Gamma = \{j \in \Lambda : \Lambda_j = false\}} \tau_{il}}, & \text{if } \Lambda_j\text{=false ;} \\ 0, & \text{otherwise.} \end{cases} \quad (3.5)$$

For each $i$, the sum of the components $p_{ij}$ is equals to one so, if $r$ is a random number uniformly distributed in [0,1], we can select the $j$-th and the $(j+1)$-th component of $\vec{p}_i$ using the following pseudo-code:

Let b=0;j=0;
**while** $b < r$ or $L_{eCITN_j} \geq i + n - 1$ **do**
  $b \leftarrow p_{ij}$; $j \leftarrow j + 1$
**end while**

It must be noted that this building process is analogous of the $\nabla^T$ operator.

- The last phase, relative to the update of the pheromone, is done in the same way of the max-min ACO inside the hyper-cube framework [Blum et al., 2001]: if we indicate with $s_{ib}$ the best solution generated in the iteration $k$ by the $h$ ants (the parameter $h$ is given by the user), with $s_{bs}$ the best-so-far solution generated since the start of the algorithm, and with $s_{rb}$ the best solution generated since the last restart of the algorithm (see [Blum et al., 2001] for details), the procedure for updating the pheromone can be divided in two part: the first part in

which we reinforce the pheromone trails of $s_{ib}$,$s_{bs}$,$s_{rb}$

$$\tau_{is_{ib_{ij}}} \leftarrow \tau_{is_{ib_{ij}}} \cdot \rho \cdot k_{ib} \quad i = 1...|I|, j = 1...2. \tag{3.6}$$

$$\tau_{is_{bs_{ij}}} \leftarrow \tau_{is_{bs_{ij}}} \cdot \rho \cdot k_{bs} \quad i = 1...|I|, j = 1...2. \tag{3.7}$$

$$\tau_{is_{rb_{ij}}} \leftarrow \tau_{is_{rb_{ij}}} \cdot \rho \cdot k_{rb} \quad i = 1...|I|, j = 1...2. \tag{3.8}$$

and the second part in which we apply the evaporation of the pheromone trails

$$\tau_{ij} \leftarrow \tau_{ij} - \rho \cdot \tau_{ij} \quad i = 1...|I|, j = 1...|I| + |E|. \tag{3.9}$$

## 3.5 Hyperheuristic models

*Hyperheuristics* are defined in the literature [Terashima-Marin et al., 1999, Hart and Ross, 1998, Fang et al., 1994, Berger et al., 1999] to be high-level heuristics which choose between heuristics in order to solve a given optimization problem. The fundamental principle exploited by a hyperheuristic is the cooperation between different research strategies (i.e., low-level knowledge-poor heuristics, or metaheuristics) finalized to obtaining better performances than the execution of the single ones. It must be noted that a hyperheuristic is not an hybrid metaheuristic: the latter takes diverse features from a set of metaheuristics and generates a new one that embodies them; the former uses a set of metaheuristics (hybrid or not) and run them in different moments of the search. An algorithm outline of a hyperheuristic is given in Figure 3.7:

**procedure** Hyperheuristic
Let $MH$ be a set of metaheuristics and mh a metaheuristic
**while** ( not `GlobalStopCriterion` ) **do**
  $mh$=`SelectNewMetaheuristic`
  `UpdateParameters(`$mh$`)`
  **while** (not `LocalStopCriterion(`$mh$`))` **do**
    `Execute(`$mh$`)`
    `ApplyFeedbackSchema(`$mh$`)`
  **end while**
**end while**

Figure 3.6: High-level pseudo-code for an hyperheuristic.

The functions needed to define it are the following:

- The function `GlobalStopCriterion`, that represents the stop criterion for the procedure Hyperheuristic.

- The function `SelectNewMetaheuristic`, that selects a metaheuristic $m \in MH$.
- The function `UpdateParameters`, that sets all the parameters and features relative to the metaheuristic $mh$.
- The function `LocalStopCriterion`, that represents the stop criterion for the execution of $mh$.
- The function `Execute`, that runs the metaheuristic selected.
- The function `ApplyFeedbackSchema`, that applies a feedback schema [Battiti and Tecchiolli, 1994] for modifying the parameters of the running metaheuristic.

It must be observed that when the set $MH$ is made of only one metaheuristic, the hyperheuristic collapses into a metaheuristic, In this work we have developed a model of hyperheuristic called *random token-ring hyperheuristic* (RTRH). We will describe each of them in the following.

### 3.5.1 Random token ring hyperheuristic

The random token-ring hyperheuristic (RTRH) is made of a set of metaheuristics $MH$ such that $|MH| > 1$. RTRH takes the attribute random because the selection of the next metaheuristic to use is done in a random way. RTRH takes the attribute token-ring because the set of metaheuristics is imagined as a ring in which each metaheuristic passes the token to another one. Our implementation of RTRH is made of four metaheuristics: tabu search, iterated local search, simulated annealing, ant colony optimization; in the following is given an algorithm outline of a random token ring hyperheuristic:

**Global stop criterion:** This is the most external loop. It is normally based on temporal criteria: we assign a temporal interval $T = [t^{'}, t'']$ to RTRH and after $t^{''}$ all computational processes must stop. RTRH divides the temporal interval $T$ into a set of $n$ subintervals $\tau_i$ such that

$$\sum_{i=1}^{n} |\tau_i| = (t^{''} - t') \tag{3.10}$$

**Local stop criterion:** RTRH assigns $\tau_i$ to the selected metaheuristic $mh \in$ **MH**. If we indicate with $q_I$ the initial value of the cost function at

**procedure** RandomTokenRingHyperheuristic
Let **MH** and **MH**$'$ be sets of metaheuristics.
Let assume initially **MH** $= \{TS, SA, ILS, ACO\}$ and **MH**$' = \emptyset$
**while** not `GlobalStopCriterion` **do**
  Select and remove randomly a metaheuristic $mh \in$ **MH**
  `UpdateParameters(`$mh$`)`
  **while** not `LocalStopCriterion(`$mh$`)` **do**
    `Execute(`$mh$`)`
    `ApplyFeedbackSchema(`$mh$`)`
  **end while**
  insert $mh$ in **MH**$'$
  **if MH** $= \emptyset$ **then**
    **MH** $=$ **MH**$'$ and **MH**$' = \emptyset$
  **end if**
**end while**

Figure 3.7: High-level pseudo-code for the random token-ring hyperheuristic (RTRH).

the begin of the time slot $\tau_i$ in which runs the metaheuristic $mh$, with $q_F$ the final value of the cost function at the end of the time slot $\tau_i$ in which has run the metaheuristic $mh$, with $n_f$ the numbers of changes in the cost function during the time slot $\tau_i$, and with $t$ the length of the time slot $\tau_i$, indicating with $\varphi$ the number:

$$\varphi = \frac{\frac{\arctan(q_I - q_F) + \frac{\pi}{2}}{\pi} + \frac{n_f}{t}}{2} \tag{3.11}$$

the local stop criterion can be written in this way:

**stop running criterion (metaheuristic m)**

$length \leftarrow length_{\tau_i}$
return $(length = length_{\tau_i} + \varphi \cdot length_{\tau_i})$

As we can see, $\varphi$ is a real number comprised in $[0, 1]$, so when $\varphi = 0$ then the length of the time slot $\tau_i$, assigned to the metaheuristic $mh$, is not increased; when $\varphi = 1$ then the length of the time slot $\tau_i$ is increased of the same length, in other words it is assigned to the metaheuristic $mh$ an other time slot having the length equal to the length of $\tau_i$.

Otherwise, if $0 < \varphi < 1$, the length of time slot $\tau_i$ is increased of $\varphi \cdot length_{\tau_i}$.

**Executing m:** if RTRH has assigned the time slot $\tau_i$ to the metaheuristic $mh \in$ {ACO,TS,ILS}, by naming with $t_0$ and $t$ respectively the lower bound and the upper bound of $\tau_i$, the parameters $\alpha$, $\beta$,$\gamma$,$\delta$ change respecting this table:

| variables | $[t_0, \frac{(t-t_0)}{4})$ | $[\frac{(t-t_0)}{4}, \frac{(t-t_0)}{2})$ | $[\frac{(t-t_0)}{2}, 3\frac{(t-t_0)}{4})$ | $[3\frac{(t-t_0)}{4}, (t-t_0))$ |
|---|---|---|---|---|
| $\alpha$ | $MAX/3$ | $MAX/4$ | 0 | 0 |
| $\beta$ | $MAX/3$ | $MAX/4$ | $MAX/4$ | 0 |
| $\gamma$ | $MAX/3$ | $MAX/4$ | $MAX/4$ | 0 |
| $\delta$ | 0 | $MAX/4$ | $MAX/2$ | $MAX$ |

(3.12)

Initially $\alpha$,$\beta$,$\gamma$,$\delta$ are set like the first column. If we don't have modification in the cost function values during the interval $[t_0, (t-t_0)/4)$, in the subsequently interval $[(t-t_0)/4, (t-t_0)/2)$ the values of $\alpha$,$\beta$,$\gamma$,$\delta$ are changed using the second column of the table, and so on. If we have at least a modification in the cost function in whatever temporal interval $t^* \in \tau_i$, this loop restarts again from the first column in the table, using the following temporal intervals: $[t^*, (t-t^*/4)]$, $[(t-t^*)/4, (t-t^*)/2]$, $[(t-t^*)/2, 3(t-t^*)/4]$, $[3(t-t^*)/4, (t-t^*)]$. If the selected metaheuristic is SA, we will use similarly the following table:

| sub time slot of $\tau_i$ | operator | a |
|---|---|---|
| $[t_0, \frac{(t-t_0)}{4})$ | $\nabla^E_\mu$ | 0.99 |
| $[\frac{(t-t_0)}{4}, \frac{(t-t_0)}{2})$ | $\nabla^E_{macro}$ | 1.1 |
| $[\frac{(t-t_0)}{2}, 3\frac{(t-t_0)}{4})$ | $\nabla^I$ | 0.99 |
| $[3\frac{(t-t_0)}{4}, (t-t_0))$ | $\nabla^T$ | 1.1 |

(3.13)

The value assigned to each variable in each time slot is assigned in according to thumb rules.

**UpdateParameters:** this procedure allows to map the state of a metaheuristic into another one. We have designed two models for RTRH:

**local restart** consisting into executing, during the time slot $\tau_i$, the selected metaheuristic $m$ as it restarted from the begin. This means: to reset the tabu list for TS, the temperature for SA, the pheromone for ACO and so on.

**mapping** consisting into mapping the state of the metaheuristic $mh'$, run in the time slot $\tau_{i-1}$, into the state of the metaheuristic $mh''$,

running in the time slot $\tau_i$. Storing the last $k$ current solutions of the time slot $\tau_{i-1}$, we can map the state of $mh^{'}$ into the state of $mh^{''}$ using this table:

| Metaheuristic | mapping |
|---|---|
| **TS** | Delete all elements stored into the tabulist. Insert in the tabu list the last $z$ distinct current solutions, with $z \leq tabulist_{length} \leq k$. |
| **SA** | Set the value of the temperature $T = \lvert\frac{f(S_{current})-f(S_{best})}{f(S_{best})}\rvert \cdot 100$. |
| **ILS** | ——— |
| **ACO** | For each of the last $k$ current solutions: add the pheromone to the components of the $i$-th current solution and apply the evaporation process. |

(3.14)

# Chapter 4

# The framework

In this chapter we will describe the framework that we have developed for solving the phylogenetic inference problem. We will begin with an overview of the framework, then we will analyze each of its components.

## 4.1 An overview

The framework is inspired to the Easy Local++ framework [Schaerf et al., 2000, Gaspero and Schaerf, 2003]. It is made up by a set of overlapped abstraction layers: *I/O layer*, *Core layer*, *Metaheuristic layer*, *Hyperheuristic layer*.

Each abstraction layer embodies cooperating components that take care of different aspects of the search. Each layer of the hierarchy relies on the services supplied by lower levels and provides a set of more abstract operations to the upper layers.

## 4.2 I/O layer

I/O layer is the lowest level of the hierarchy and it is responsible for the communication with the external environment by getting the input that has to be analyzed and delivering the output representing the best solution found by the framework. It is made of two component: *input system* (IS) and the *output system* (OS).

The input system is responsible for interfacing the framework with any input source: XML files, nexus files, data bases and so on. The structure of the Input system is modular, so if we need to read a new type of input source, it is sufficient to add the appropriate method. All type of input is internally transformed by the Input system into two structures: the DNA matrix, that

provides the respective DNA sequence for each taxon analyzed, and the taxa names list that provides the respective name for each taxon analyzed.

The output system is responsible for interfacing the framework with any output source: XML files, nexus tree format and so on. Also the structure of the Output system is modular, so if the international community defines a "standard tree format" for the representation of a phylogeny, it is sufficient to add the respective method to the output system. The output system transforms the internal data structures of the Core layer in the standard nexus tree format, and write down the output in a XML file.

## 4.3 Core layer

The core layer embodies the components that represent specific aspects of a local search: *state manager*, that represents the state of the search; *engine*, that represents the manager of the neighborhood; *evaluator*, that represents the manager of the cost function. All the metaheuristics use these components, for reading the state, for generating new solutions, for evaluating new solutions.

The state manager supplies a set of operations for managing the state of the search. It is responsible for: storing into the memory the solution structures (best so far solution, current solution and so on); storing the value of each solution structure (best so far value, current solution value and so on); storing the values of the constant values used for the neighborhood exploration.

The engine supplies a set of methods implementing: the neighborhood operators ($\nabla^E_\mu$, $\nabla^E_{macro}$, $\nabla^I$, $\nabla^T$), and the initial solution generator. It stores the structure representing the neighborhood of a solution as an array of solutions of which only the first element is used by the simulated annealing, while the whole array is used by the other three metaheuristics, representing the ants for aco, the neighborhood for tabu search and iterated local search.

The evaluator provides a set of methods which handle the cost function. It is responsible for: storing the pre-computed matrix $\mathbf{\Omega}$ and the dynamic matrix $\mathbf{\Psi}$, storing all the models for the cost matrix $\mathbf{C}$, implementing the operators $\|\cdot\|_{min}$, $\oplus$ and $\circlearrowleft$.

Each component of the core can use any other component inside the core, for example: the engine can use the state manager for computing the neighborhood of a solution; the evaluator can use the neighborhood generated by the engine for computing the best solution; the state manager can use the evaluator for setting up the best so far solution, the current solution and so on.

We can say, using only a sentence, that the core is made up by *fundamental, cooperating and universal components*. Fundamentals because they are necessary for searching in the solutions space. Cooperating because they cooperate during the search. Universal because they can be used indistinctly by any metaheuristic.

## 4.4 Metaheuristic layer

The metaheuristic layer embodies all the metaheuristics used by the framework. These metaheuristics are represented as complex components, are characterized by an high abstraction level, and are totally independent from the representation of the problem, adhering to the philosophy used by Gaspero and Schaerf [2003].

The metaheuristics implemented are components extended by an abstracted class called `metaheuristic`. This class is characterized by the generic methods `RUN()`, and `SelectBest()`. All the metaheuristics must implement these methods, so they provide to the highest layer a common interface for managing any of the metaheuristic.

A component embodies subcomponent, for example: the metaheuristic ACO embodies the class that manages the hypercube; respectively for the tabu search with its tabu list and the simulated annealing with the annealing model.

## 4.5 Hyperheuristic layer

The highest layer of the framework is the hyperheuristic layer, representing the level in which we describe a generic hyperheuristic. Each describes: how many metaheuristics and which one have to participate, in which order they have to be called, how much times to dedicate to each metaheuristic, when a metaheuristic in runtime must stop, which strategy to apply during the runtime of a metaheuristic. The framework includes the description of the two hyperheuristics: the simple hyperheuristic and the random token ring hyperheuristic. Both of them were described in the prevoius chapter.

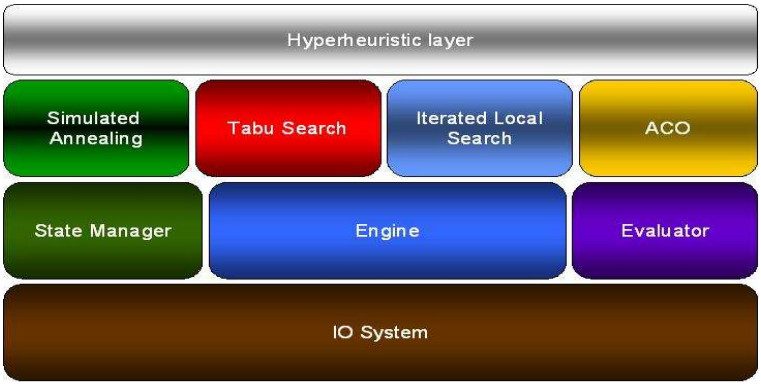


Figure 4.1: The framework.

# Chapter 5

# Results

We tested the proposed hyperheuristics and metaheuristics on a test set made of one hundred different instances, each of them containing fifty DNA sequences. Each DNA sequence is made of one thousand characters, generated randomly. For each instance we have fixed the initial solution using the deterministic procedure in Figure 5.1. All instances, the random procedure for generating them, and the deterministic procedure for generating the initial solution can be found at the internet site:
*http://iridia.ulb.ac.be/∼dcatanzaro/BioProject-Eufonìa*.

The cost matrix used for the cost function is the Wagner Model (see Swofford et al. [1996]).

The combination of strategies has produced 22 different implementations of hyperheuristic and metaheuristic models[1]:

Table 5.1: Table of Experiments

| Hyperheuristics | Metaheuristics | Neighborhood operators | Updating parameters |
|---|---|---|---|
| — | ACO | $\nabla^E_{macro}$, $\nabla^E_{\mu}$, $\nabla^I$, $\nabla^T$,mix | ——— |
| — | ILS | $\nabla^E_{macro}$, $\nabla^E_{\mu}$, $\nabla^I$, $\nabla^T$,mix | ——— |
| — | SA | $\nabla^E_{macro}$, $\nabla^E_{\mu}$, $\nabla^I$, $\nabla^T$,mix | ——— |
| — | TS | $\nabla^E_{macro}$, $\nabla^E_{\mu}$, $\nabla^I$, $\nabla^T$,mix | ——— |
| RTRH | ACO,ILS,SA,TS | mix | local restart |
| RTRH | ACO,ILS,SA,TS | mix | mapping |

We have tested these strategies using the time as evaluation criterion: we have run the strategies for 1000 seconds over each instance, using a cluster of six Athlon XP 1400+, each of them equipped with 512 MB of RAM, Debian linux distribution version 3.0 with kernel version 2.4.26, and gcc compiler version 2.95.4.

[1] In the table mix indicates "all operators".

```
procedure GenerateInitialSolution
let M ∈ M^(ℝ)_{|I|×(|E|∪|I|)}
c ← 0
for i = 1 to number of taxa−1 do
  for j = 1 to 2 do
    M_ij = c
    c ← c + 1
  end for
end for
```

Figure 5.1: Pseudo-code for the deterministic generation of the initial solution.

## 5.1 Time-based analysis

In Fig. 5.2 we report, using the box-and-whisker plot [Siegel and Castellan, 1988], the absolute values of the solutions generated by each hyperheuristic/metaheuristic. A box shows the range between the 25% and the 75% quantile of the data. The median of the data is indicate by a bar. The whiskers extend to the most extreme data point which is no more than 1.5 times the interquantile range from the box. Outliers are indicate as circles.

Wishing to make a comparison between the hyperheuristics and the metaheuristics described (from now on we will call them simply algorithms), we have to determine a zero value and an interval of measurement [Johnson, 2002]. Instance by instance we can assign the zero value to the algorithm that found the best value, and the value one to the the algorithm that found the worst value. Each value generated by each algorithm in each instance can be re-mapped using this formula:

$$\frac{x_j - min_i}{max_i - min_i} \quad \forall i \in Instances, \forall j \in Algorithms \tag{5.1}$$

where $x_i$ is the best value found by the algorithm $j$, $min_i$ and $max_i$ are respectively the best and the worst value found running all the algorithms on the instance $i$. Working in this way we obtain the box-and-whisker plot histograms in Fig. 5.3, in which we represent the scaled values of the solutions generated by each algorithm and their relative ranks in the comparison among each other. Analyzing these histograms we can observe that the group of metaheuristics using SA obtains the best performances followed by RTRH with mapping and RTRH with local restart. Analyzing these histograms for all type of operators we obtain the histograms in Fig. 5.4, 5.5, 5.6, 5.7 and 5.8. We can observe:

- The SA metaheuristic obtains always the best performances.

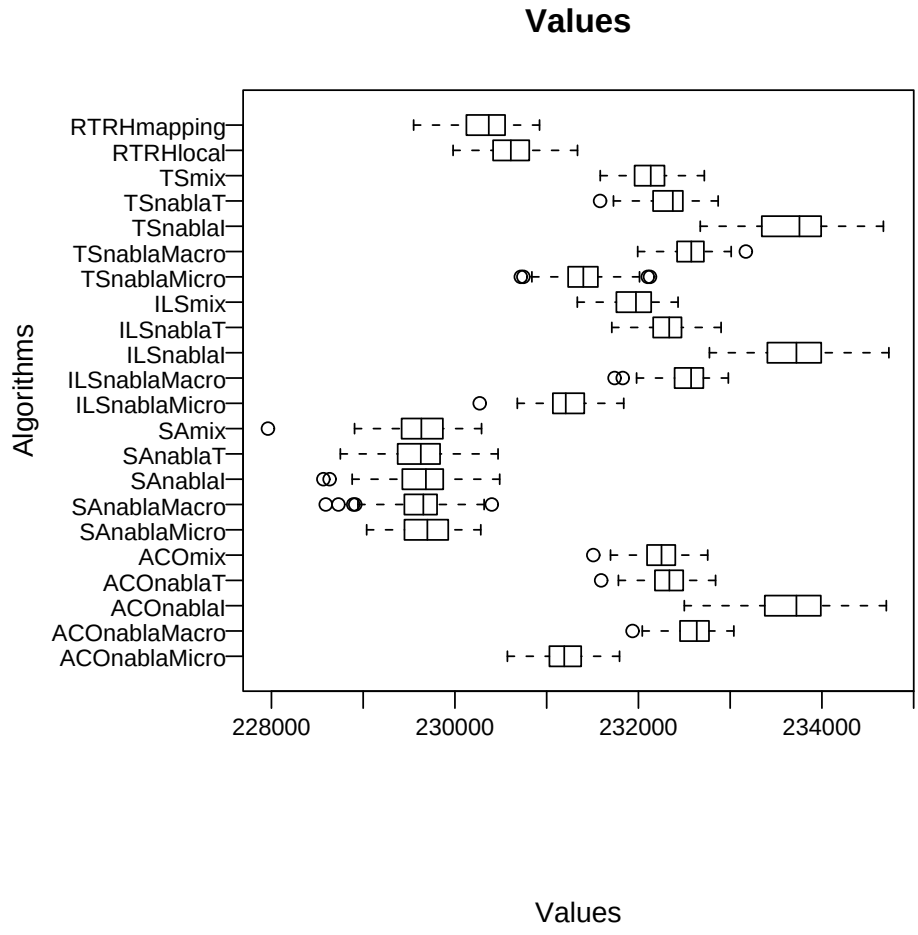


Figure 5.2: Global box-and-whisker plot histogram of experiments using the time as stop criterion. A box shows the range between the 25% and the 75% quantile of the data. The median of the data is indicate by a bar. The whiskers extend to the most extreme data point which is no more than 1.5 times the interquantile range from the box. Extreme points are indicate as circles.

- The ACO metaheuristic is the second best algorithm with respect to the operator $\nabla^E_\mu$ (ACOnablaMicro) while ILSnablaMicro and TSnablaMicro are respectively the third and the forth.

- ILSnablaMacro and TSnablaMacro are the second best metaheuristics with respect to the operator $\nabla^E_{macro}$, while ACOnablaMacro is the last in terms of performance.

- Executing the paired Wilcoxon rank sum test [Siegel and Castellan, 1988] (see table 5.1) we can affirm that all the algorithms using the operator $\nabla^I$ are statistically equivalents, and in particular obtain the global worst performances.

- With respect to the operator $\nabla^T$ the paired Wilcoxon test allows to say with confidence level 95% and with Holmes correction for multiple tests, that ACOnablaT and ILSnablaT are statistically equivalents and are the second best models in terms of performances, while TSnablaT is the last.

- Using all the operators, the second best performances are obtained by ILSmix, while TSmix is the third and ACO the last in term of performances.

Grouping the algorithms described by the type of metaheuristic used we can observe:

**ACO:** The best performance is obtained (Fig. 5.9) exploring the neighborhood by using the $\nabla^E_\mu$ operator, the second one using all operators, the third one using $\nabla^T$ operator, the fourth one using $\nabla^E_{macro}$ operator, and the last one using $\nabla^I$ operator.

**ILS:** The best performances is obtained (Fig. 5.10) exploring the neighborhood by using the $\nabla^E_\mu$ operator, the second one using the mix strategy, the third one using $\nabla^T$ operator, the fourth one $\nabla^E_{macro}$ operator, and the last one using $\nabla^I$ operator.

**SA:** Executing the paired Wilcoxon rank sum test it is possible to observe how all the operators give results statistically equals.

**TS:** The best performances is obtained (Fig. 5.12) exploring the neighborhood by using the $\nabla^E_\mu$ operator, the second one using the mix strategy, the third one using $\nabla^T$ operator, the fourth one $\nabla^E_{macro}$ operator, and the last one using $\nabla^I$ operator.

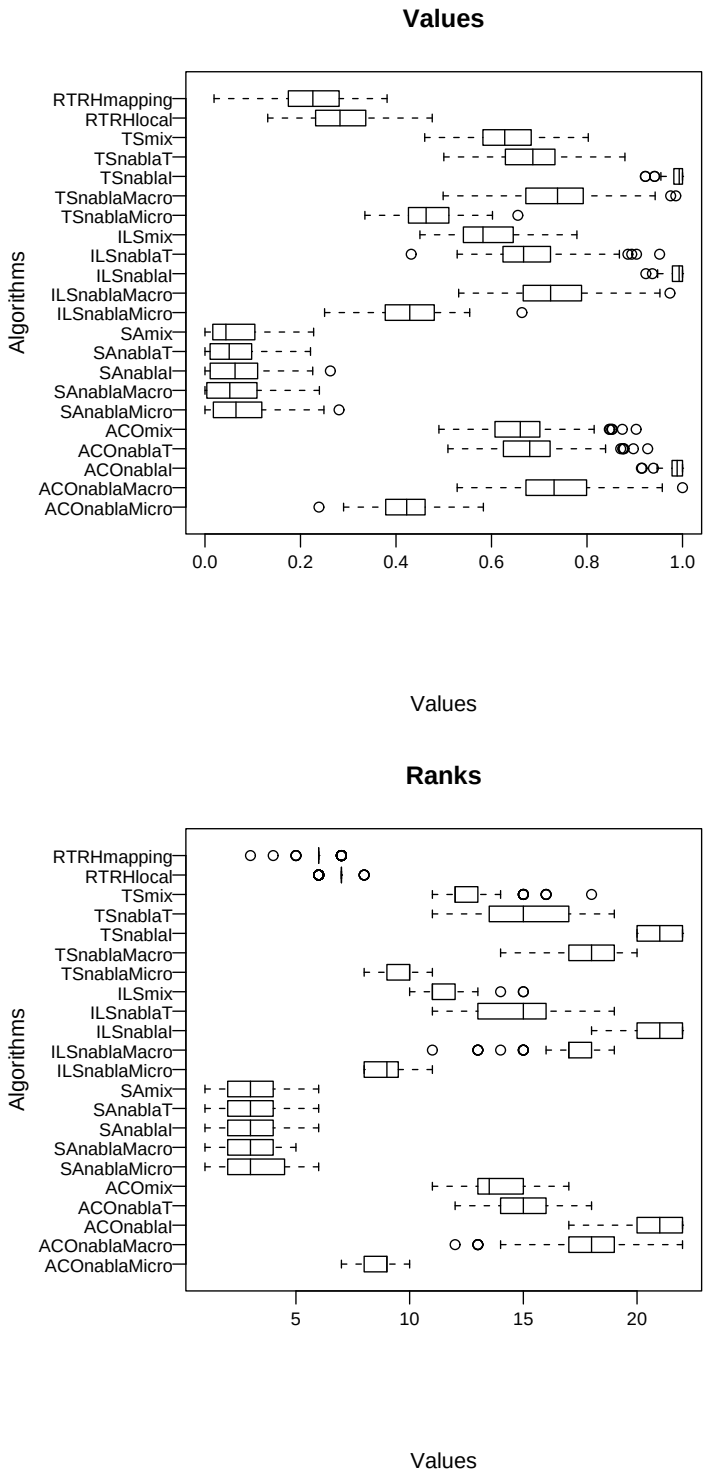


Figure 5.3: The scaled values of the solutions generated by each algorithm (top) and their relative ranks in the comparison among each other (bottom) are depicted in two box-and-whisker plot histogram.

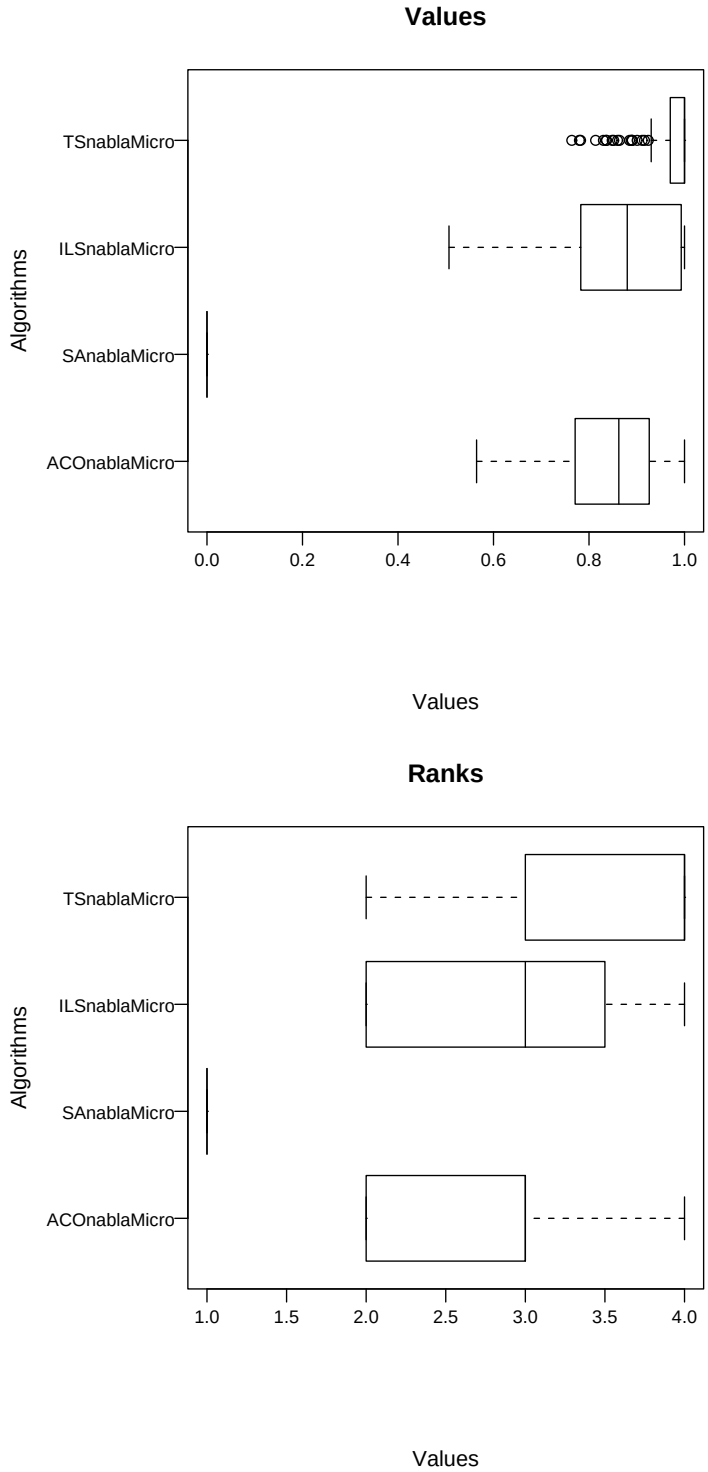


Figure 5.4: Box-and-whisker plot histogram relative to all the hyperheuristics using $\nabla_{\mu}^{E}$ as neighborhood operator. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

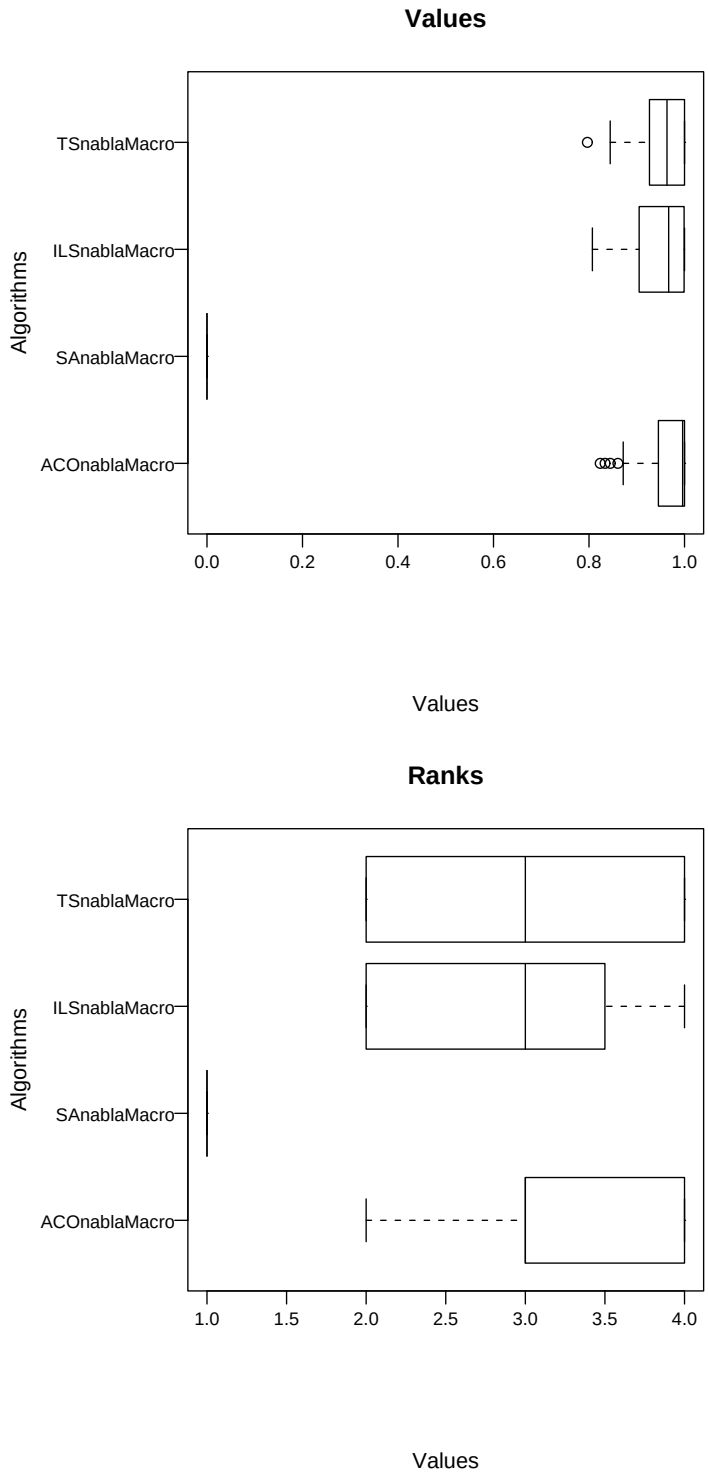


Figure 5.5: Box-and-whisker plot histogram relative to all the hyperheuristics using $\nabla^{E}_{macro}$ as neighborhood operator. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

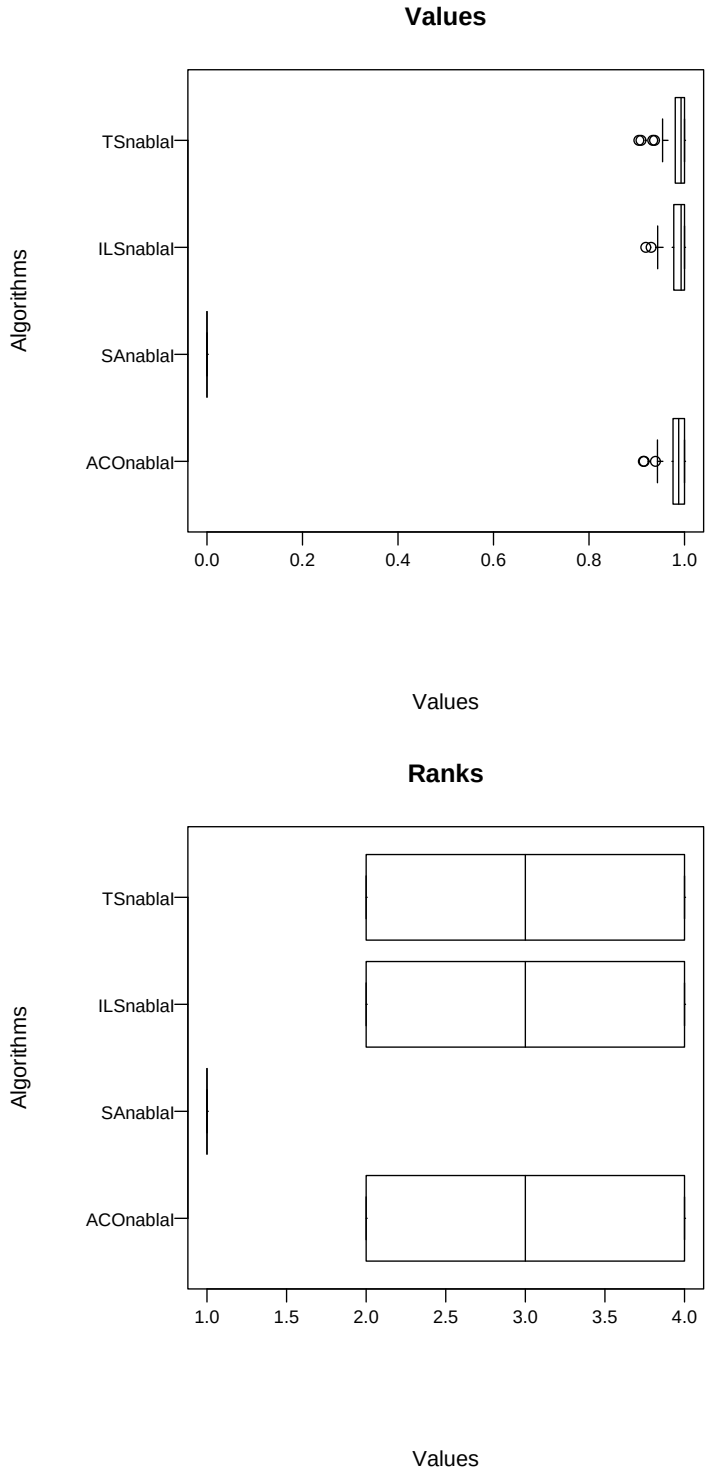


Figure 5.6: Box-and-whisker plot histogram relative to all the hyperheuristics using $\nabla^I$ as neighborhood operator. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

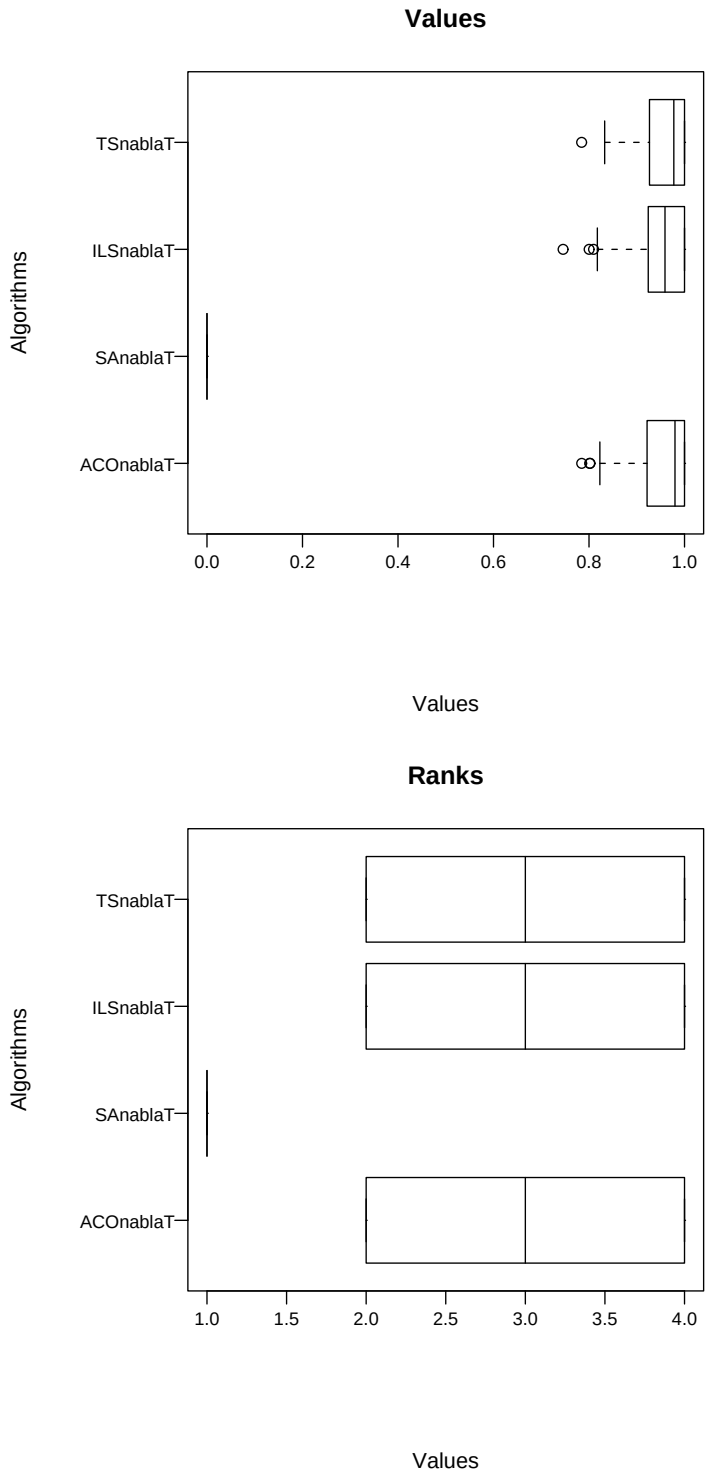


Figure 5.7: Box-and-whisker plot histogram relative to all the algorithms using $\nabla^T$ as neighborhood operator. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

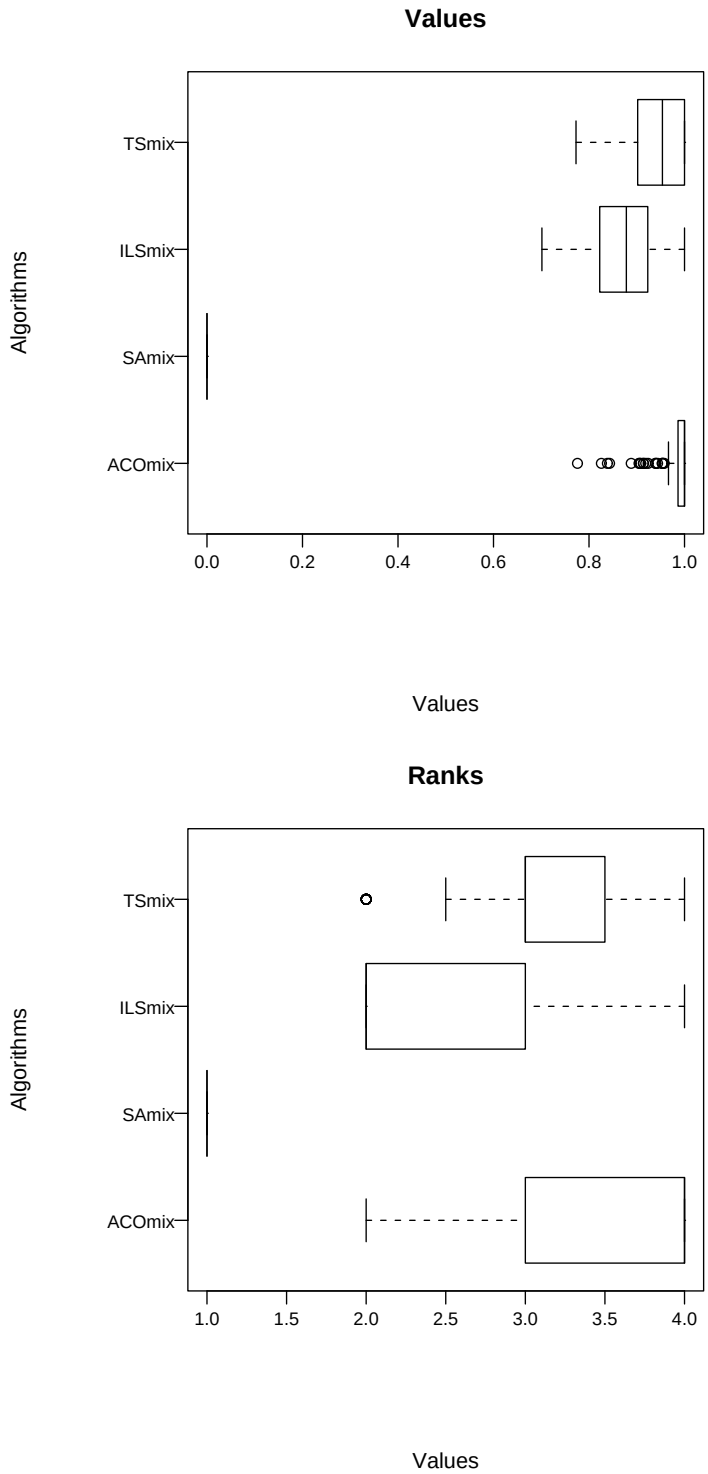


Figure 5.8: Box-and-whisker plot histogram relative to all the algorithms using all the neighborhood operators. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

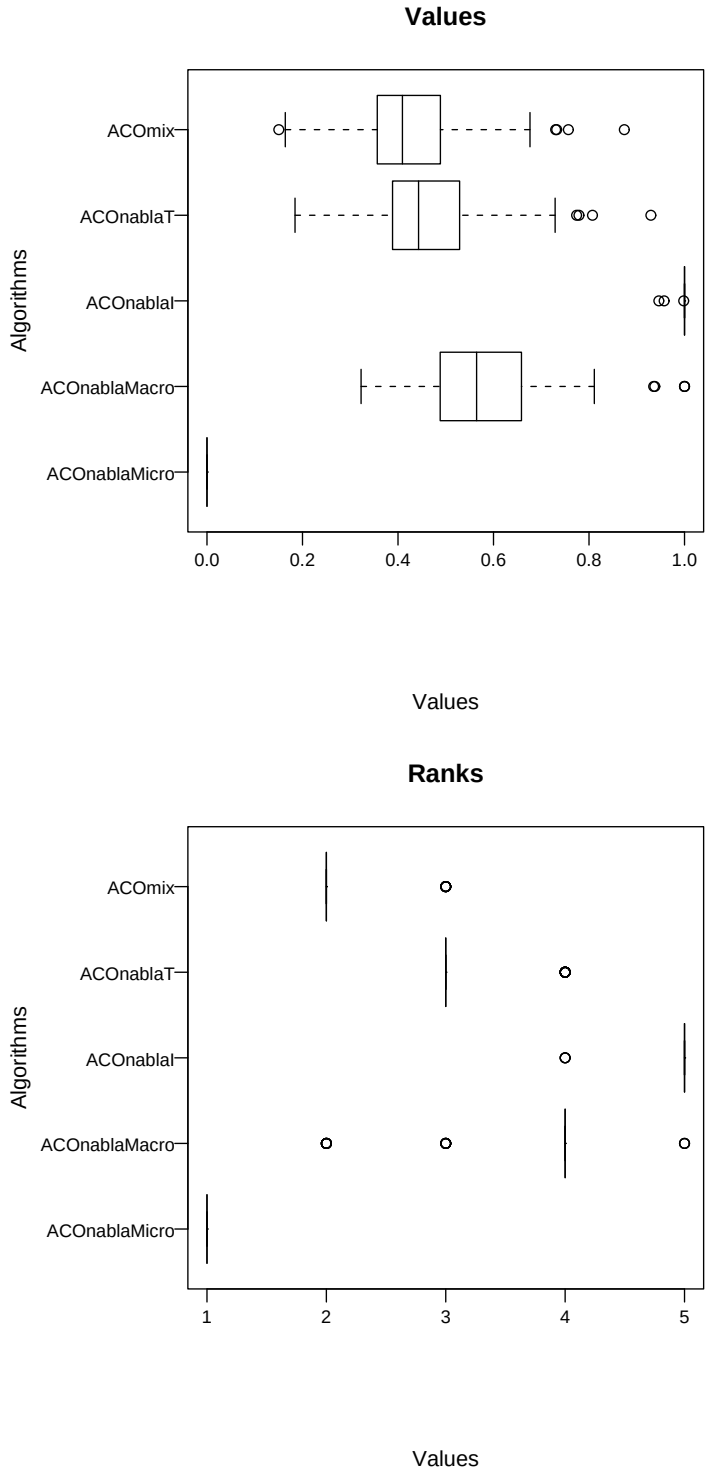


Figure 5.9: Box-and-whisker plot histogram relative to the group of algorithms using ACO as metaheuristic. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

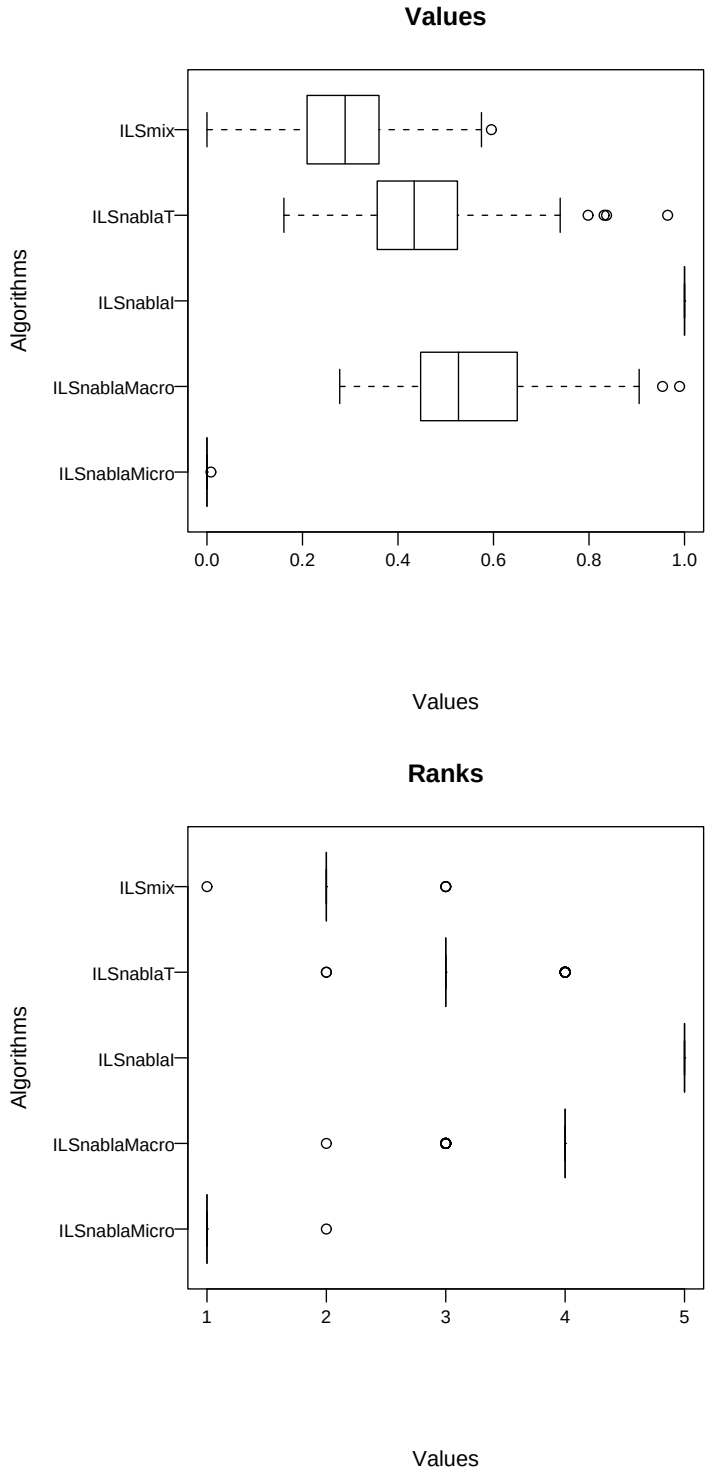


Figure 5.10: Box-and-whisker plot histogram relative to the group of algorithms using ILS as metaheuristic. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

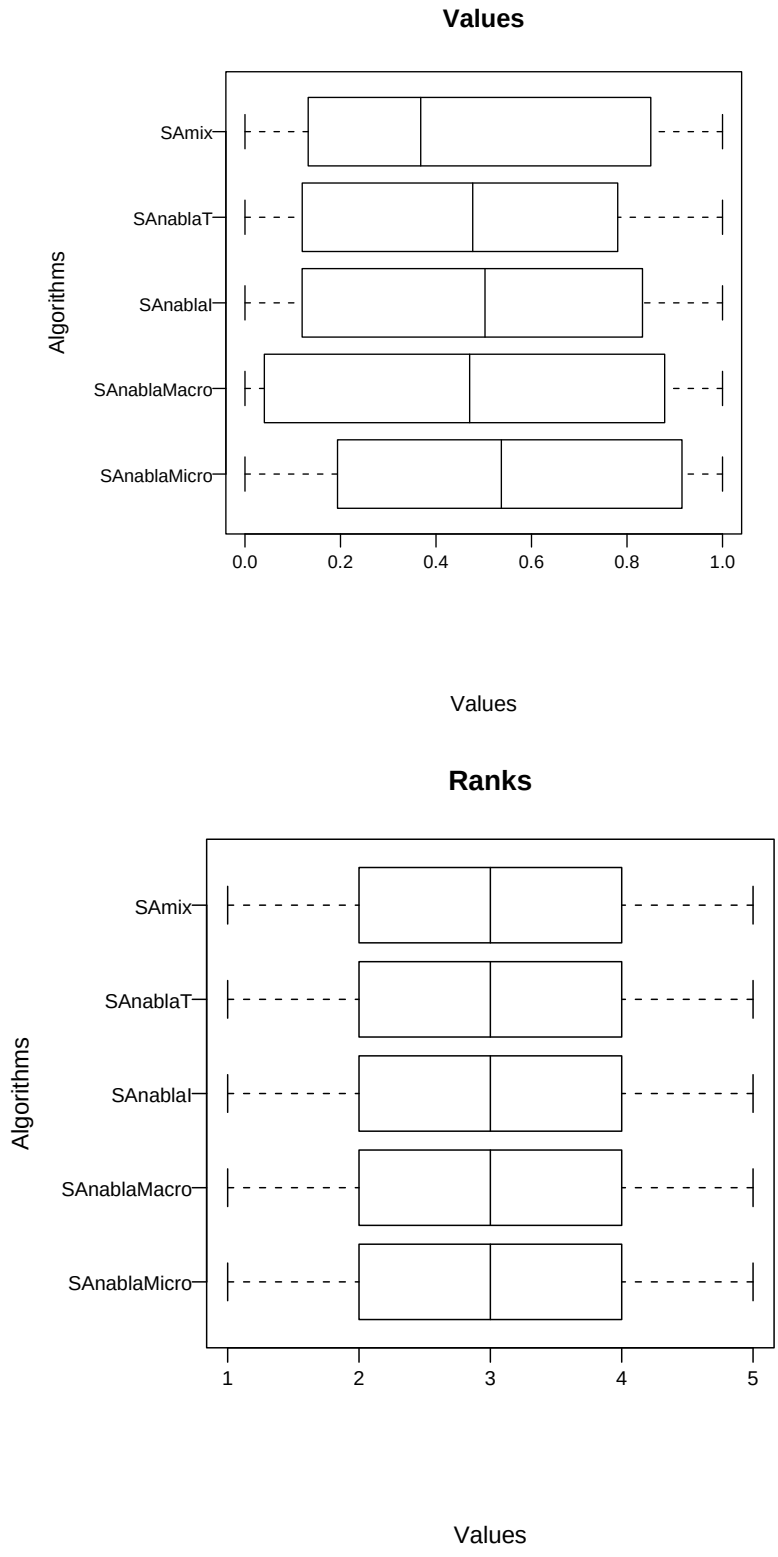


Figure 5.11: Box-and-whisker plot histogram relative to the group of algorithms using SA as metaheuristic. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

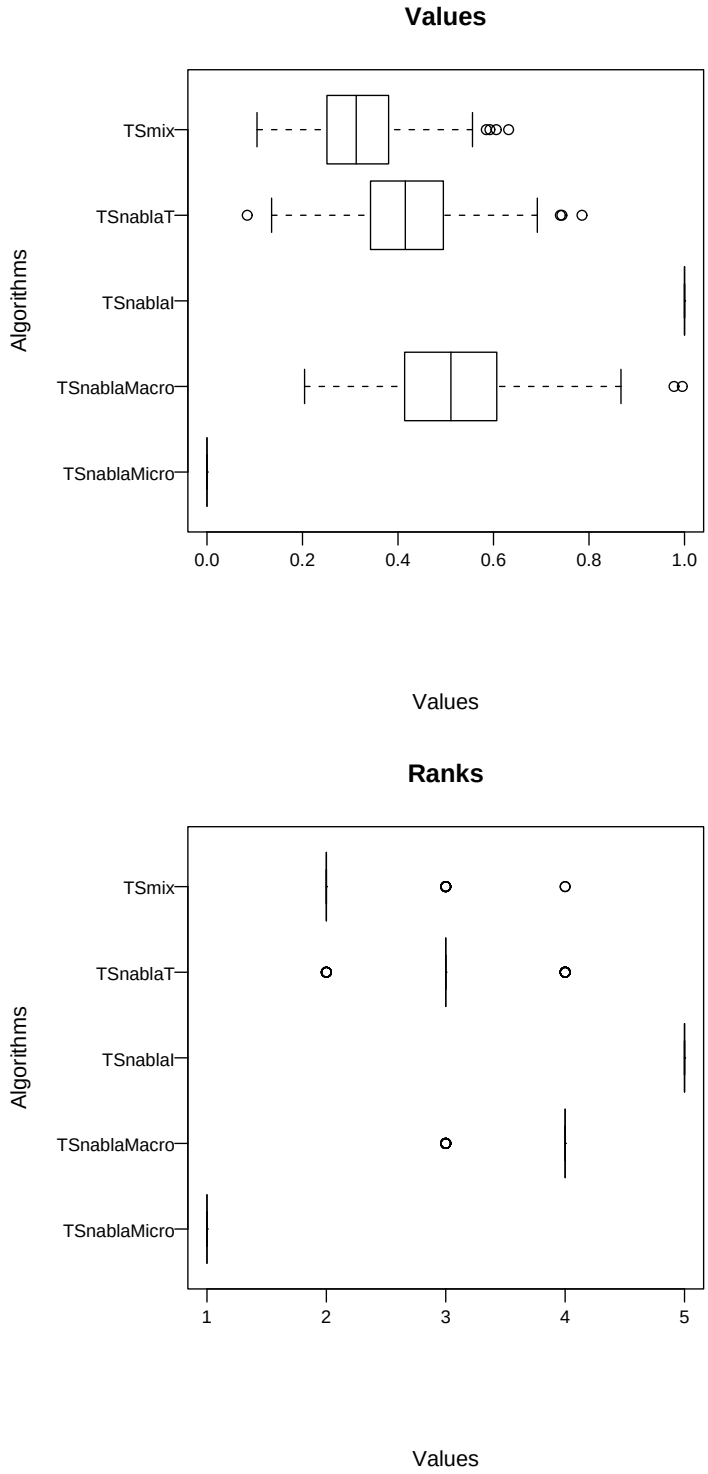


Figure 5.12: Box-and-whisker plot histogram relative to the group of algorithms using TS as metaheuristic. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

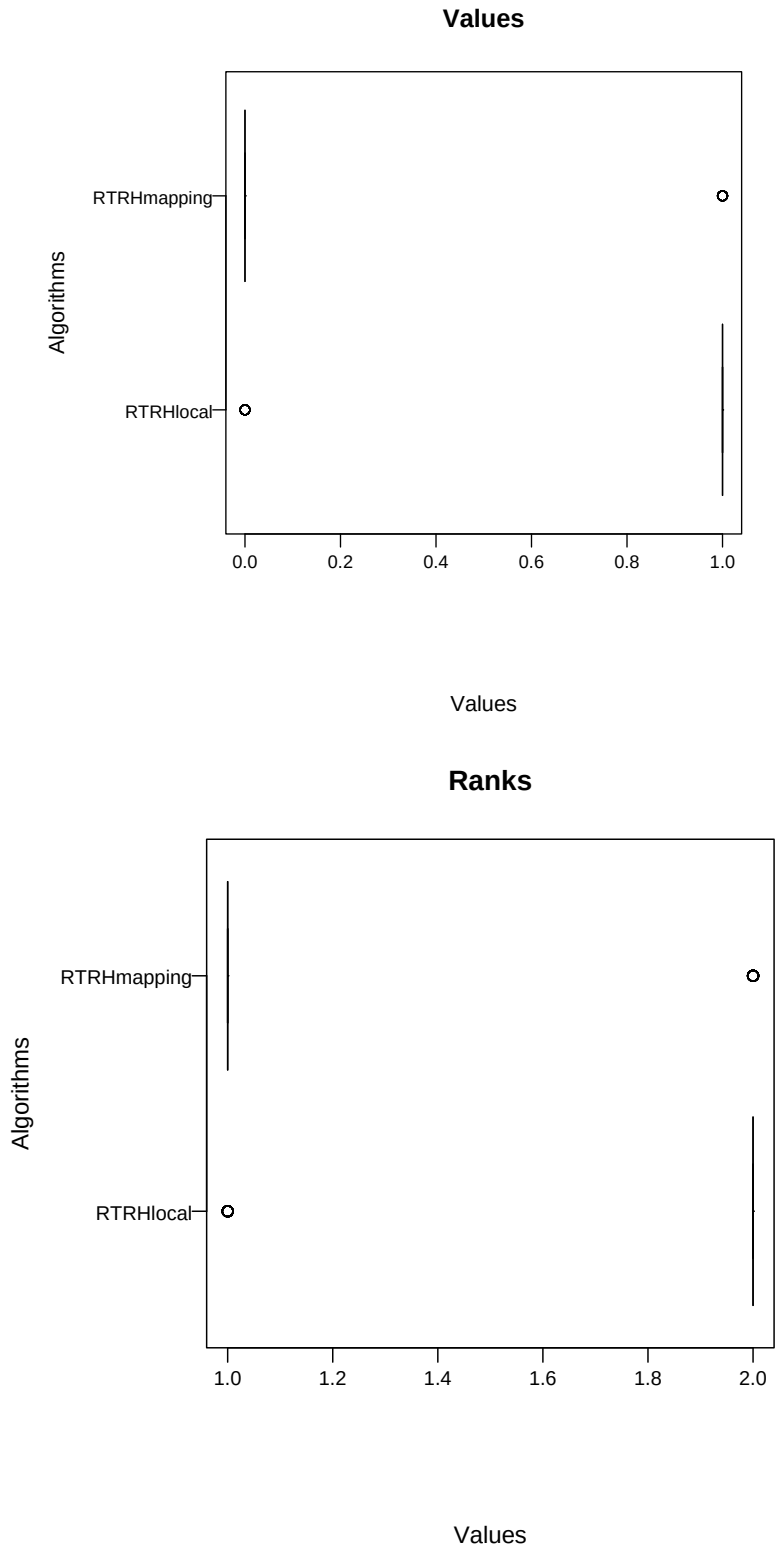


Figure 5.13: Box-and-whisker plot histogram relative to the RTRH group of algorithms. The scaled values of the solutions generated by each algorithm (left) and their relative ranks in the comparison among each other (right) are depicted in two box-and-whisker plot histogram.

Grouping the RTRH hyperheuristic models (Fig. 5.13) we can observe that RTRH with mapping obtains performances higher that RTRH with local restart. In other words, to proceed with remapping the state of a metaheuristic in an analogous one of another metaheuristic is better than restarting a metaheuristic from the begin.

## 5.2 Conclusions

Taking into account the above results, we can affirm that:

- the Simulated Annealing metaheuristic obtains the best results over these set of instances; referring to the type of operator, the best median of the solutions found is relative to the mix strategy, although statistically there's no difference among the results obtained using the other operators, as we can see from Figure 5.3 and 5.11.
- The RTRH family performs in average not worst than the other metaheuristics. In particular, as we can see from Figure 5.3, RTRH family are the second and third algorithm in terms of results. This is a good result if we think that, at this moment, does not exist a policy for managing the metaheuristics different from the random one, where for policy we mean a set of criteria for selecting, running and stopping a metaheuristic. Anyway, we can observe this important result: mapping the state of a metaheuristic into an analogous one of another metaheuristic, during the passage from one metaheuristic to another one, performs better than starting the new metaheuristic from the start.
- Using the data structure for representing a tree and matrices for computing the quality function allows to halve the computational cost of the Sankoff's algorithm, as we showed in the Chapter 2.

## 5.3 Future work

The hyperheuristic models open a new possible way for searching the solutions spaces of the combinatorial optimization problems. The way for combining a subset of metaheuristics, selecting the metaheuristic for the temporal run time, selecting the stop criterion are limited only by the imagination of the researcher. We will focus our future research on these topics, trying to describe the criteria for which a hyperheuristic could find the optimal performances for a given set of instances.

| | ACOmix | ACOnablaI | ACOnablaMacro | ACOnablaMicro | ACOnablaT | ILSmix | ILSnablaI | ILSnablaMacro | ILSnablaMicro | ILSnablaT |
|---|---|---|---|---|---|---|---|---|---|---|
| ACOnablaI | 0.000e+00 | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| ACOnablaMacro | 0.000e+00 | 6.355e-16 | NA | NA | NA | NA | NA | NA | NA | NA |
| ACOnablaMicro | 6.355e-16 | 6.355e-16 | 6.355e-16 | NA | NA | NA | NA | NA | NA | NA |
| ACOnablaT | 0.000e+00 | 6.355e-16 | 6.149e-13 | 0.000e+00 | NA | NA | NA | NA | NA | NA |
| ILSmix | 5.441e-13 | 6.355e-16 | 6.355e-16 | 0.000e+00 | 4.240e-15 | NA | NA | NA | NA | NA |
| ILSnablaI | 0.000e+00 | 1.000e+00 | 0.000e+00 | 0.000e+00 | 0.000e+00 | 0.000e+00 | NA | NA | NA | NA |
| ILSnablaMacro | 9.103e-15 | 6.355e-16 | 5.129e-01 | 0.000e+00 | 1.449e-11 | 0.000e+00 | 6.355e-16 | NA | NA | NA |
| ILSnablaMicro | 6.355e-16 | 6.355e-16 | 6.355e-16 | 1.000e+00 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | NA | NA |
| ILSnablaT | 1.026e-01 | 6.355e-16 | 1.008e-12 | 0.000e+00 | 1.000e+00 | 0.000e+00 | 6.355e-16 | 1.947e-12 | 0.000e+00 | NA |
| RTRHlr | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 |
| RTRHmap | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 |
| SAmix | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 |
| SAnablaI | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 |
| SAnablaMacro | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 |
| SAnablaMicro | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 |
| SAnablaT | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 |
| TSmix | 1.265e-04 | 6.355e-16 | 6.355e-16 | 0.000e+00 | 4.153e-10 | 1.966e-09 | 6.355e-16 | 6.355e-16 | 0.000e+00 | 6.664e-09 |
| TSnablaI | 0.000e+00 | 1.000e+00 | 0.000e+00 | 0.000e+00 | 0.000e+00 | 0.000e+00 | 1.000e+00 | 0.000e+00 | 0.000e+00 | 0.000e+00 |
| TSnablaMacro | 0.000e+00 | 6.355e-16 | 1.000e+00 | 0.000e+00 | 1.334e-12 | 0.000e+00 | 6.355e-16 | 1.000e+00 | 0.000e+00 | 1.598e-13 |
| TSnablaMicro | 6.355e-16 | 6.355e-16 | 6.355e-16 | 1.338e-09 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 6.355e-16 | 7.887e-07 | 6.355e-16 |
| TSnablaT | 2.345e-03 | 6.355e-16 | 1.516e-11 | 0.000e+00 | 1.000e+00 | 0.000e+00 | 6.355e-16 | 1.357e-10 | 0.000e+00 | 1.000e+00 |

| | RTRHlr | RTRHmap | SAmix | SAnablaI | SAnablaMacro | SAnablaMicro | SAnablaT | TSmix | TSnablaI | TSnablaMacro | TSnablaMicro |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ACOnablaI | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| ACOnablaMacro | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| ACOnablaMicro | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| ACOnablaT | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| ILSmix | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| ILSnablaI | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| ILSnablaMacro | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| ILSnablaMicro | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| ILSnablaT | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| RTRHlr | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| RTRHmap | 1.298e-08 | NA | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| SAmix | 6.355e-16 | 6.355e-16 | NA | NA | NA | NA | NA | NA | NA | NA | NA |
| SAnablaI | 6.355e-16 | 6.355e-16 | 1 | NA | NA | NA | NA | NA | NA | NA | NA |
| SAnablaMacro | 6.355e-16 | 6.355e-16 | 1 | 1 | NA | NA | NA | NA | NA | NA | NA |
| SAnablaMicro | 6.355e-16 | 6.355e-16 | 1 | 1 | 1 | NA | NA | NA | NA | NA | NA |
| SAnablaT | 6.355e-16 | 6.355e-16 | 1 | 1 | 1 | 1 | NA | NA | NA | NA | NA |
| TSmix | 0.000e+00 | 0.000e+00 | 0 | 0 | 0 | 0 | 0 | NA | NA | NA | NA |
| TSnablaI | 0.000e+00 | 0.000e+00 | 0 | 0 | 0 | 0 | 0 | 0.000e+00 | NA | NA | NA |
| TSnablaMacro | 0.000e+00 | 0.000e+00 | 0 | 0 | 0 | 0 | 0 | 0.000e+00 | 6.355e-16 | NA | NA |
| TSnablaMicro | 0.000e+00 | 0.000e+00 | 0 | 0 | 0 | 0 | 0 | 6.355e-16 | 6.355e-16 | 6.355e-16 | NA |
| TSnablaT | 0.000e+00 | 0.000e+00 | 0 | 0 | 0 | 0 | 0 | 7.126e-12 | 6.355e-16 | 1.915e-11 | 0 |

Table 5.1: Paired Wilcoxon rank sum test.

# Bibliography

J. Adachi and M. Hasegawa. Molphy: programs for molecular phylogenetics based on maximum likelihood. Technical Report 27, Institute of Statistical Mathematics, Tokyo, Japan, 1992.

R. Agarwala and D. Fernández-Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM Journal on Computing*, 23(6):1216–1224, 1994.

R. K. Ahuja, J. B. Orlin, and D. Sharma. Multi-exchange neighborhood structures for the capacitated minimum spanning tree problem. *Mathematical Programming*, 91:71–97, 2001.

R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126– 140, 1994.

A. D. Baxevanis and B. F. Francis Ouellette. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins.* John Wiley and Sons Inc., New York, NY, 2001.

J. Berger, M. Sassi, and M. Salois. A hybrid genetic algorithm for the vehicle routing problem with time windows and itinerary constraints. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 44–51, Orlando, Florida, USA, 1999.

C. Blum, A. Roli, and M. Dorigo. HC–ACO: The hyper-cube framework for Ant Colony Optimization. In *Proceedings of MIC'2001 – Metaheuristics International Conference*, volume 2, pages 399–403, Porto, Portugal, 2001.

M. Dorigo and T. Stützle. *Ant Colony Optimization.* MIT Press, Cambridge, MA, 2004.

A. W. M. Dress. Phylogenetic combinatorics. Please contact directly the author at dress@mis.mpg.de for obtaining this work, 2000.

A. W. M. Dress. Recent results and new problems in phylogenetic combinatorics. Please contact directly the author at dress@mis.mpg.de for obtaining this work, 2001.

H-L. Fang, P. M. Ross, and D. Corne. A promising hybrid ga/heuristic approach for open-shop scheduling problems. In John Wiley and Sons Ltd, editors, *Proceedings of ECAI 94: 11th European Conference on Artificial Intelligence*, pages 590–594. Cohn Editor, Boston, MA, USA, 1994.

J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Sunderland, UK, 2002.

M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman, NY, USA, 2003.

L. Di Gaspero and A. Schaerf. Easylocal++: An object-oriented framework for flexible design of local search algorithms. *Software Practice and Experience*, (33)8:733–765, 2003.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 5:533–549, 1986.

F. Glover and G. A. Kochenberger. *Handbook of Metaheuristics*. Kluwer Academic Publishers, Boston, MA, USA, 2003.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Boston, MA, 1997.

D. Gusfield. The steiner tree problem in phylogeny. Technical Report 334, Yale University, New Haven, CT, 1984.

E. Hart and P. M. Ross. A heuristic combination method for solving jopshop scheduling problems. In A. E. Eiben, T. Back, M. Schoenauer, and H-P.Schwefel, editors, *Parallel Problem Solving from Nature V, eds.*, pages 845–854. Springer-Verlag, Berlin, Germany, 1998.

J. P. Huelsenbeck. Performance of phylogenetic method in simulation. *Syst.Biol.*, 44:17–48, 1995.

D. S. Johnson. A theoretician's guide to the experimental analysis of the algorithms. In M. Goldwasser, D. S. Johnson, and C. C. McGeoch, editors, *Proceedings of the 5th and 6th DIMACS Implementation Challenges,*. American Mathematical Society, NY, USA, 2002.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search. In F. Glover and G. A. Kochenberger, editors, *Handbook of Metaheuristics*. Kluwer Academic Publisher, Boston, MA, USA, 2002. to appear.

N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21: 1087–1092, 1953.

G. J. Olsen, H. Matsuda, R. Hagstrom, and R. Overbeek. Fastdnaml: a tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Computer Applications in the Biosciences*, 10(1):41–48, 1994.

C. Papadimitriou and K. Steiglitz. *Combinatorial Optimization, Algorithm and Complexity*. Dover Publications, Mineola, NY, USA, 1998.

I. Pe'er, R. Shamir, and R. Sharan. Incomplete directed perfect phylogeny. *SIAM Journal on Computing*, 33(3):590–607, 2004.

T. H. Reijmers, R. Wehrens, F. D. Daeyaert, P. J. Lewi, and L. M. C. Buydens. Using genetic algorithms for the construction of the phylogenetic trees: application to g-protein coupled receptor sequences. *BioSystem*, 49: 31–43, 1999.

F. S. Roberts and L. Sheng. Phylogeny numbers for graphs with two triangles. *Discrete Applied Mathematics*, 103:191–207, 2000.

K. H. Rosen. *Discrete and Combinatorial Mathematics*. CRC Press, FL, USA, 2000.

A. Rzhetsky and M. Nei. A simple method for estimating and testing minimum evolution trees. *Comput. Appl. Biosci.*, 10:409–412, 1992.

M. P. Scaparra, S. Pallottino, and M. G. Scutellà. Large scale local search heuristics for the capacitated vertex p-center problem. *Networks*, 91:1–11, 2003.

A. Schaerf, M. Cadoli, and M. Lenzerini. Local++: A c++ framework for local search algorithms. *Software Practice and Experience*, 30(3):233–256, 2000.

R. Shamir and D. Naor. Algorithms for molecular biology. Technical Report 8, School of computer science, Tel Aviv University, Tel Aviv, Israel, 2002.

S. Siegel and N.J. Castellan. *Nonparametric statistics for the behavioral sciences*. Mc Graw Hill, Singapore, Indonesia, 1988. Second edition.

A. Stamatakis and T. Ludwig. Accelerating parallel maximum likelihood-based phylogenetic tree calculations using subtree equality vectors. In *In Proceedings of 15th IEEE/ACM Supercomputing Conference (SC2002), Proceedings on CD*, Baltimore, Maryland, USA, 2002.

A. Stamatakis and T. Ludwig. Phylogenetic tree inference on pc architectures with axml/paxml. In *In Proceedings of 17th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS2003), High Performance Computational Biology Workshop*, Nice, France, 2003.

A. Stamatakis, T. Ludwig, and H. Meier. A fast program for phylogenetic tree inference with maximum likelihood. In *To be published in Proceedings of 2nd Joint HLRB and KONWIHR Result and Reviewing Workshop*, Munich, Germany, 2004a.

A. Stamatakis, T. Ludwig, and H. Meier. New fast and accurate heuristics for inference of large phylogenetic trees. In *In Proceedings of 18th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS2004), High Performance Computational Biology Workshop, Proceedings on CD,*, Santa Fe, New Mexico, 2004b.

M. Steel, M. D. Hendy, and D. Penny. Reconstructing phylogenies from nucleotide pattern probabilities: A survey and some new results. *Discrete Applied Mathematics*, 88:367–396, 1998.

K. Strimmer and A. von Haeseler. A quartet puzzling: a quartet maximum likelihood method for reconstructing tree topologies. *Mol. Biol. Evol.*, 13: 964–969, 1996.

D. L. Swofford, G. J. Olsen, P. J. Waddell, and D. M. Hillis. *Phylogenetic inference in molecular systematics*. Sinauer Associates, Sunderland, UK, 1996.

H. Terashima-Marin, P. M. Ross, and M. Valenzuela-Rendon. Evolution of constraint satisfaction strategies in examination timetabling. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-99)*, pages 635–642, Orlando, Florida, USA, 1999.