Advanced methods to solve the maximum parsimony problem

Karla Esmeralda Vazquez Ortiz

Thèse de Doctorat

Karla E. VAZQUEZ ORTIZ

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université d'Angers
sous le label de l'Université de Nantes Angers Le Mans

École doctorale : Sciences et technologies de l'information, et mathématiques

Discipline : Informatique, section CNU 27
Unité de recherche : Laboratoire d'etudes et de recherche en informatique d'Angers (LERIA)

Soutenue le 14 Juin 2016
Thèse n° : 77896

Advanced methods to solve the maximum parsimony problem

JURY

Présidente :	**M^me^ Christine SINOQUET**, Maître de conférences, HDR, Université de Nantes
Rapporteurs :	**M. Cyril FONTLUP**, Professeur des universités, Université du Littoral
	M. El-Ghazali TALBI, Professeur des universités, INRIA Lille Nord Europe
Examinateur :	**M. Thomas SCHIEX**, Directeur de Recherche INRA, HDR,
Directeur de thèse :	**M. David LESAINT**, Professeur des universités, Université d'Angers
Co-directeur de thèse :	**M. Jean-Michel RICHER**, Maître de conférences, Université d'Angers

Acknowledgments

This work was carried out with the economic support of CONACyT and technical and administrative support of the LERIA laboratory of the University of Angers. I want to express them my sincere gratitude. Also I want to address my thanks to my thesis committee: Cyril Fontlup, El-Ghazali Talbi, Thomas Schiex, Christine Sinoquet for their assistance and their reviews. Most of all I am indebted to Pr. David Lesaint my director for his understanding and his patience.

I want to say thank you to Dr. Eduardo Rodriguez Tello, for his contributions in this research, for his advice and for motivating me to get my PhD.

Many thanks to my husband Jean-Michel for being with me during my research as my advisor and my understanding husband, for supporting me when I most needed it. We finished the thesis but we will continue the life together with a little angel called Sarah that we could add to our phylogenetic or genealogical tree.

My children in some way gave me the chance to continue my career, they represent a big source of motivation to succeed in life, I want to tell them thank you for their understanding and for continuing supporting me despite the distance that separates us, I love you.

Thank you to the persons of my family who helped me in each travel to Mexico, like my aunts Conchita and Lupita Vazquez, and my mother who took good care of my treasures Emily and Vanessa. Thanks to all who gave motivation words and those who told me I would not be able to succeed, they gave me a reason to make more efforts and I could prove them they were wrong.

Arturo Avalos Benitez a great person I got to know in the last years, who, without knowing me and without being part of my family became a very important support for me and my daughters during all this time, thank you for everything that you have made for us.

To my son Adrian, if some day you read this I want you to know that you are a very important part of this and of my life.

Thank you God, you built a wonderful destiny for me and in the difficult moments I could feel your presence.

Contents

List of Tables

List of Figures

1

Introduction

1.1 Background

Bioinformatics is an interdisciplinary research area where intervene: biology, statistics, applied mathematics, chemistry, biochemistry, artificial intelligence and computational science. Bioinformatics studies the development of computational methods and statistical techniques to solve practical and theoretical problems derived from the storage, extraction, handling and distribution of information related with biological macromolecules such as DNA (Deoxyribonucleic acid), RNA (Ribonucleic acid) and proteins. These macromolecules are represented by their primary structure [1]. Hereafter the term *"biological macromolecules"* will be used to refer to DNA, RNA and protein sequences.

In Bioinformatics there are two important fields. The first one develops databases and computational tools. The second one uses these databases and tools to extract biological knowledge in order to understand the living systems. It is important to note that these fields are complementary and one can benefit from the other.

Some of the main research areas in bioinformatics are:

— **Sequence alignment:** this refers to the comparison procedure of two or more sequences of macromolecules by looking for a series of individual characters or character patterns that are in the same order in the sequences to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Identical or similar characters are placed in the same column, and non-identical characters can either be placed in the same column as a mismatch or opposite to a gap [2] in one of the other sequences. In an alignment, non-identical characters and gaps are placed so as to bring as many identical or similar characters as possible into columns.

— **Protein structure prediction:** is the prediction of the three-dimensional structure of a protein from its amino acid sequence i.e., the prediction of its secondary [3], tertiary [4], and quaternary structure from its primary structure. It is very important in bioinformatics and theoretical chemistry and it is critical in medicine (for example in drug design) and biotechnology (for example in the design of new enzymes).

— **Gene prediction:** is the problem of locating genes in a genomic sequence [5], i.e. identifying the regions of genomic

1. *The primary structure* of a protein is the linear sequence of its amino acid structural units, and partly comprises its overall biomolecular structure.
2. *Gap*: a special character used to indent columns and that serves as a spacer.
3. *Secondary structure*: is the local spatial arrangement of a polypeptide's backbone atoms regardless of the conformations of its side chains.
4. *Tertiary structure*: refers to the three-dimensional structure of an entire polypeptide.
5. *Genomic sequence*, is the entire nucleic acid sequence necessary for the synthesis of a functional protein (or functional RNA). It includes regions that contain no information.

DNA that encode genes. It includes protein-coding genes as well as RNA genes, but may also include prediction of other functional elements such as regulatory regions. Gene finding is one of the first and most important steps in understanding the genome of a species once it has been sequenced.

— **Prediction of genic expression:** is the process by which information from a gene is used in the synthesis of a functional gene product. These products are often proteins, but in non-protein coding genes such as ribosomal RNA (rRNA), transfer RNA (tRNA) or small nuclear RNA (snRNA) genes, the product is a functional RNA. The process of gene expression is used by eukaryotes and prokaryotes to generate the macromolecular machinery for life. Several steps in the gene expression process may be modulated, including the transcription, RNA splicing, translation, and post-translational modification of a protein.

— **Protein-protein interactions:** occurs when two or more proteins bind together, often to carry out their biological function. Many of the most important molecular processes in the cell such as DNA replication are carried out by large molecular machines that are built from a large number of protein components organized by their protein-protein interactions. As protein-protein interactions are so important there are a multitude of methods to detect them (Phizicky and Fields, 1995).

— **Evolution modeling:** evolution is the change in the inherited characteristics of biological populations over successive generations. Evolutionary processes give rise to diversity at every level of biological organization, including species, individual organisms and molecules such as DNA and proteins. Evolution modeling tries to model the changes that occur in biological populations.

— **Reconstruction of phylogenetic trees:** a phylogenetic tree or evolutionary tree is a branching diagram or *tree* showing the inferred evolutionary relationships among various biological species or other entities based upon similarities and differences in their physical and/or genetic characteristics. The sequences (also called taxa, singular taxon) joined together in the tree are considered to descend from a common ancestor. The phylogenetic reconstruction tries to model the evolutionary relationships between a group of species in structures of trees (rooted or unrooted) using their genetic information represented by sequences of characters.

In the remaining of this work, we will focus on phylogenetic reconstruction also called Phylogenetics.

There are two main approaches used to the phylogenetic reconstruction. The first one known as comparative morphology, is based on the analysis of anatomic, physiologic and behavioral characteristics shared between the studied organisms. The second one, called comparative biochemistry, studies the common characters (amino acids and nucleotides) of macromolecules that belong to a group of analyzed individuals.

Different methods have been designed for the reconstruction of phylogenetic trees, Maximum Parsimony (MP) is one of the most interesting in the field of combinatorial optimization. MP is a reconstruction method based on the assumption that the most probable hypothesis is the one that requires the fewest explanations. MP is used to classify the living beings with the purpose of recovering their phylogenetic history. According to Cladism [6], the species should be classified according to monophyletic groups [7] (Hennig, 1966). MP tries to maximize the evolutionary similitudes between species, this implies identifying a tree topology with the minimum *tree length*, i.e. the topology that requires the smallest number of evolutionary changes (or transformation in characters state) in order to explain the differences between the sequences (Kluge and Farris, 1969).

1.2 Motivation

There are two main reasons for the study of phylogenetic tree reconstruction. The first one concerns biology where it is used for the generation of new vaccines, antibacterial, herbicide molecules and in the study of the dynamic of microbial communities (Pace, 1997). For example, in the pharmaceutical industry, it helps to the development of new drugs. It is worth mentioning, that in molecular biology, MP is used in the prediction of protein structures, determination of homology of sequences and the classification of proteins (Murakami and Jones, 2006).

The second reason is related to Computer Science where MP is a NP-complete problem (Garey and Johnson, 1977), due to its equivalence to the problem of combinatorial optimization known as *The problem of the Steiner tree in a hypercube* which represents an important challenge due to its nature and the fact that the solutions are not vectors of values but trees.

6. *Cladism*: biology area that defines the evolutionary relationships between organisms based on derived similitudes.
7. A *Monophyletic group* is composed by the common ancestor and all its descendants.

Indeed, the size of the search space, i.e. the number of possible rooted tree topologies, for this problem is given by the next expression (Xiong, 2006):

$$\frac{(2n-3)!}{2^{n-2}(n-2)!} \tag{1.1}$$

where n is the number of studied species. It is easy to observe that this expression grows in factorial way according to the value of n, and generates huge search spaces.

The main goal of our work is to develop algorithms that provide solutions of good quality in a reasonable amount of time.

It is also worth mentioning that the cladistic community has contributed to the resolution of MP during several years and has used techniques that are well-known from the combinatorial optimization community. Those techniques were tailored to the resolution of MP.

2

State of the art

2.1 History

The phylogenetics analysis finds its origin from the XVIII century, when evolutionary theories arise and used the laws of inheritance as a fundamental basis. For this reason, an historical review with important facts will be shown.

Stephen Jay Gould argues that the French biologist Jean-Baptiste Lamarck (1744 - 1829) was the *primary evolutionary theorist* and his ideas and the way in which he structured his theory is the base of the most subsequent thinking in evolutionary biology, through to the present day (Gould, 2002). Lamarck constructed one of the first theoretical frameworks of organic evolution which was rejected during his lifetime (Burkhardt and Wellington, 1995).

Lamarck believed in the inheritance of acquired characteristics and the use and disuse model by which organisms developed their characteristics. He incorporated this belief into his theory of evolution giving rise to what we know as Lamarckism (or Lamarckian inheritance).

Lamarckism is the idea that an organism can pass on characteristics that it acquired during its lifetime to its offspring (also known as heritability of acquired characteristics or soft inheritance). Lamarck incorporated two ideas into his theory of evolution which in his days were considered to be generally true (Packard, 1901):

— *use and disuse*: individuals lose characteristics they do not require (or use) and develop characteristics that are useful.

— *inheritance of acquired traits*: individuals inherit the traits of their ancestors.

When Charles Darwin (1809 - 1882) published his theory of evolution by natural selection in *On the Origin of Species* (Darwin, 1859), he gave credence to what Lamarck called *use and disuse inheritance*, but rejected other aspects of Lamarck's theories. He proposed the mechanisms that had given origin to the whole biodiversity. His work gave rise to the *Theory of Evolution* (Darwin, 1872). The ideas of Darwin are generally summed up in the sentence *survival of the fittest*, i.e., those that have adaptation abilities to the environment will have more chance to survive, and then reproduce. Their offspring will inherit their characters. In a nutshell we can say that, from Darwin's point of view, all living species have been created from previous generations, given that all the living beings share the same genetic code. We could conclude that all the inhabitants in the planet are descendants of a common ancestor (Darwin, 1859).

Gregor Mendel (1822 - 1884) made some experiments in a small garden, which consisted of crossing plants that should differ in at least two characters. After some time of observations, he proposed the laws of inheritance. In the year 1865 he published his results with the title *Experiments on Plant Hybridization* (Mendel, 1865). Nevertheless, his work had not been recognized until the year 1900 when the scientific community began to take an interest in the mechanisms of genetic transmission.

Ernst Haeckel (1834 - 1919) in the year 1866 supported by the Darwin idea, radically defended *The Recapitulation theory*, also called the biogenetic law. It is a biological hypothesis that in developing from embryo to adult, animals go through stages resembling or representing successive stages in the evolution of their remote ancestors. This theory is currently discarded (Haeckel,

1866). He anticipated the fact that the key of the hereditary factors reside in the cell core. He was the first one to establish the phylogenetic hypothesis of the biologic biodiversity adjusted to the evolution theory.

The German entomologist Willi Hennig (1913 - 1976) gave birth to the *cladistic* analysis referring it as *phylogenetic systematics* (Hennig, 1966). The use of the terms *Cladistics* and *clade* was popularized by other researchers.

Cladistics is a classification approach in which items are grouped together based on whether or not they have one or more shared characteristics that come from the group's common ancestor and are not present in more distant ancestors. Therefore, members of the same group are thought to share a common history and are considered to be more closely related (Pearsall and Hanks, 1998; Lagassé, Goldman, Hobson, and Norton, 2000; Shaban-Nejad and Haarslev, 2008).

From the time of his original formulation until the end of the 1980s cladistics remained a minority approach to both phylogenetics and taxonomy. However, in the 1990s it rapidly became the dominant set of methods of phylogenetics in evolutionary biology, because computers made it possible to process large quantities of data about organisms and their characteristics.

Cladism is a method used to reconstruct the genealogy of organisms and is based on three principles:

1. a sequence is called a taxon (plural taxa) and represents taxonomic units in the biological system of classification of organisms,
2. every group is a descendant of a common ancestor (i.e. are monophyletic),
3. the tree with the highest probability to be the correct is the one that proves to be the most simple and has the minimum number of changes and is defined as the *most parsimonious* tree.

The Maximum Parsimony (MP) problem, i.e. the search for a tree that is most parsimonious, arises from the necessity to classify the different species, in order to exhibit their evolutionary history. Phylogenetic trees represent the possibility of representation of the evolutionary history of a group of species under the parsimony criterion.

Please note that there are different parsimony optimality criteria known as *Fitch, Wagner, Camin-Sokal, Dollo or weighted* parsimony. Those criteria determine the number of changes of a substitution from one site to another (Felsenstein, 1981):

— *Fitch* also called unweighted or unordered parsimony (Fitch, 1971) is a generalized form of Wagner parsimony where any mutation has the same cost of 1, this is the parsimony criterion that we are using in this thesis,

— *Wagner* or ordered parsimony (Wagner, 1961) imposes some order on the mutations: characters are allowed to reverse so that change from A to C costs the same number of steps as C to A, but also characters are additive so that if A to C costs 1, and C to G costs 1, then A to G must cost 2,

— *Dollo* parsimony (Farris, 1977) is based on the probability that a reversal is higher than a forward change, for example A to C is allowed with a higher probability than C to A,

— *Camin-Sokal* parsimony (Camin and Sokal, 1965) requires that all character evolution is irreversible: A to C is possible but not C to A,

— finally *generalized* parsimony (Swofford, 1990) assigns a cost to each transformation between states.

For this work, we are only interested in *Fitch* parsimony for which all substitutions are given the same weight of one unit.

Edwards and Sforza (1963) were the first ones to mention the maximum parsimony criterion in Phylogeny. They introduced computational methods to compute evolutionary trees based on genetic information. Their most important contribution was the introduction of stochastic methods of estimation applied to stochastic models of evolution.

We have analyzed the historical events that were the basis to this research problem. In the next section, the methods used to solve the phylogenetic reconstruction problem (or phylogeny) will be explained.

2.2 Methods to solve phylogenetic reconstruction

Since the 60's they all are dependent of a preliminary operation called a multiple alignment of sequences. However, as pointed out before, some software like Clustal (Thompson, Gibson, and Higgins, 2002) use distance methods with the objective to determine the order in which the sequences will be added to compose the multiple alignment.

Many methods have been developed for the phylogenetic tree reconstruction. The principal approaches are:

2.2.1 Distance based or Clustering methods

They rely on a measure of distance between species (Felsenstein, 2004), generally the Hamming distance is used. It corresponds to the number of differences between two sequences. Distance methods start by generating a first matrix that describes the

distances between each pair of sequences. Then an iterative procedure groups the two closest sequences and the matrix distance is reduced and recomputed.

The complexity of distance methods is low in the order of $O(n^2)$ which makes them suitable to analyze huge sets of data.

1. **Grouping based**

— **WPGMA (Weighted Pair Group Method with Averaging):** is a clustering method that uses a sequential clustering algorithm, in which a tree is built in a stepwise manner, by grouping sequences or groups of sequences (taxa) that are most similar to each other; i.e. for which the evolutionary distance is smallest. When two taxa are grouped, they are treated as a new single taxon (see Figure 2.1) and from among the new group of taxa, it is necessary again to identify the pair for which the similarity is the highest, and so on, until only two taxa remain (see Figure 2.1, below).

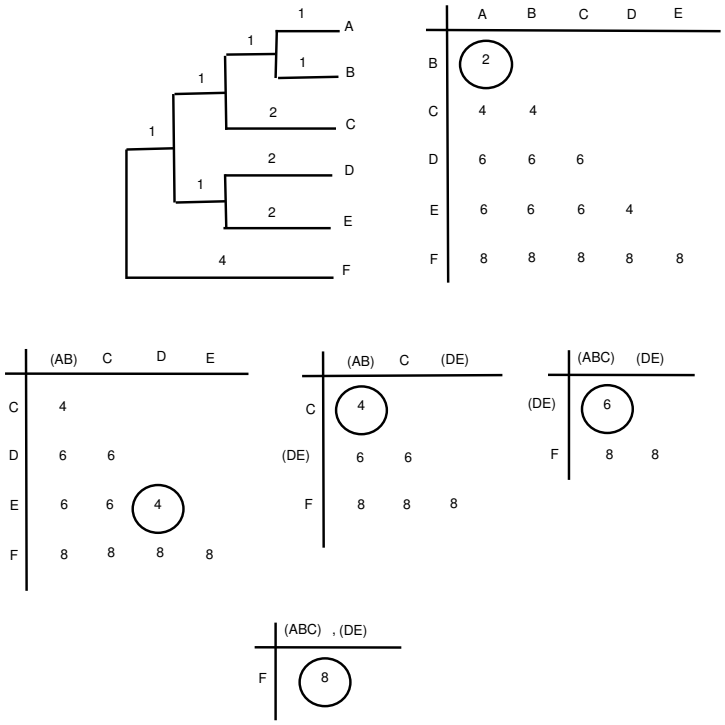


Figure 2.1 – Example of WPGMA Method

In WPGMA, the averaging of the distances is not based on the total number of taxa in the respective clusters. For example, when taxa A, B (which have been grouped before) and C are grouped into a new node 'u', then the distance from node 'u' to any other node 'k' (e.g. grouping D and E) is simply computed as follows:

$$d(u,k) = \frac{d(A,B) + d(C,k)}{2} \tag{2.1}$$

— **UPGMA (Unweighted Pair Group Method Using Arithmetic Average):** Sokal and Michener (1958) proposed this iterative process. It is similar to WPGMA where the equation to compute the new distance is:

$$d(u,k) = \frac{|AB|d(A,B) + |C|d(C,k)}{|AB| + |C|} \tag{2.2}$$

where $|C|$ is the number of taxa in C.

— **Neighbor-Joining:** is a clustering method for the creation of phylogenetic trees, created by Saitou and Nei (1987). Usually used for trees based on DNA or protein sequence data, the algorithm requires knowledge of the distance

between each pair of taxa to build the tree. In Figure 2.2(A) we can see the starting with a star tree; then a matrix Q is calculated in order to choose a pair of nodes for joining, in this case f and g. They are joined as a newly created node u (see Figure 2.2(B)). The part of the tree shown as dotted lines is now fixed and will not be changed in subsequent joining steps. The distances from node u to the nodes $a - e$ are computed. This process is then repeated, using a matrix of just the distances between the nodes, a, b, c, d, e and u, and a Q matrix derived from it. In this case u and e are joined to the newly created v, as shown in Figure 2.2(C). Two more iterations lead first to (D), and then to (E), at which point the algorithm is done, as the tree is fully resolved.

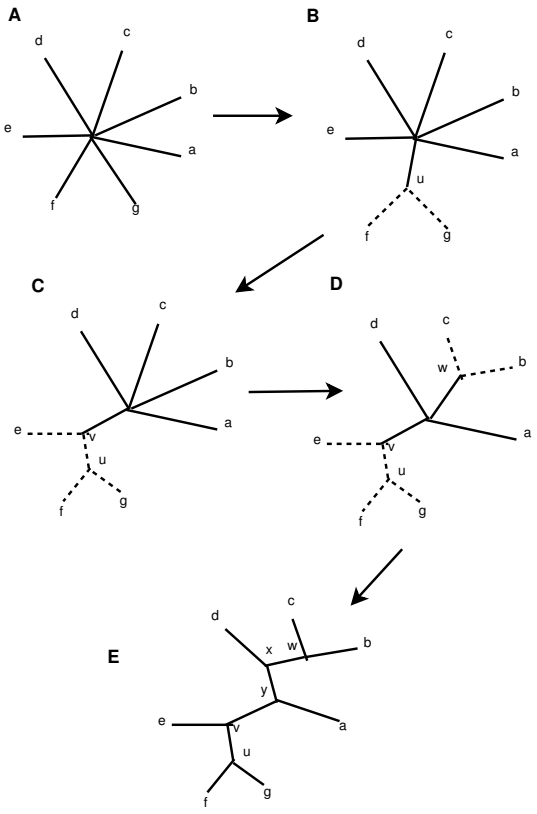


Figure 2.2 – Example of Neighbor-Joinig Method

2. **Optimality based**

   — Fitch and Margoliash (1967) proposed a method that selects the best tree taking into account the minimum deviation between the distances computed for all the branches in the tree and the distances of the set of original data.
   — **Minimum evolution**: it employs linear computation to build trees. This method searches a tree with the minimum number of mutations. Rzhetsky and Nei (1993) applied this method for the reconstruction of phylogenetic trees.

2.2.2 Characters based methods

— **Maximum parsimony:** also called *minimum-steps* or *minimum evolutionary steps* method (Felsenstein, 1973) is a character-based approach that searches the tree with the minimum number of mutations (i.e. it needs to minimize the total number of evolutionary steps), which explains in a better way the evolutionary relationships between the studied taxa represented on the leaves. MP uses a matrix of discrete phylogenetic characters to infer one or more optimal phylogenetic trees for a set of taxa and operates by evaluating candidate phylogenetic trees according to an optimality criterion. Occam's razor, a principle of theoretical parsimony suggested by William of Ockham in the 1320's, asserted that it is vain to give an explanation which involves more assumptions than necessary. Under this principle the best phylogenetic tree is the one that contains the least evolutionary changes to explain observed data (Fitch, 1971).

Trees are scored (evaluated) by using a simple algorithm to determine how many "steps" (number of mutations) are required to explain the distribution of each character. A step represents a change from one character state to another. The algorithm does not assign particular character states to internal nodes (hypothetical ancestors) on a tree: the least number of steps can involve multiple, equally costly assignments and distributions of evolutionary transitions. MP optimizes the total number of changes.

— **Maximum likelihood:** evaluates a hypothesis about evolutionary history in terms of the probability that the proposed model and the hypothesized history would give rise to the observed data set. The maximum likelihood method uses standard statistical techniques for inferring probability distributions to assign probabilities to particular possible phylogenetic trees. The method requires a substitution model to assess the probability of particular mutations; roughly, a tree that requires more mutations at interior nodes to explain the observed phylogeny will be assessed as having a lower probability.
This is broadly similar to the maximum-parsimony method, but maximum likelihood allows additional statistical flexibility by permitting varying rates of evolution across both lineages and sites. Maximum likelihood is thus well suited to the analysis of distantly related sequences, but because it formally requires search of all possible combinations of tree topology and branch length, it is computationally expensive to perform on more than a few sequences. The “pruning” algorithm, a variant of dynamic programming, is often used to reduce the search space by efficiently calculating the likelihood of subtrees (Felsenstein, 2004). The method calculates the likelihood for each site in a "linear" manner, starting at a node whose only descendants are leaves (that is, the tips of the tree) and working backwards toward the "bottom" node in nested sets. Searching tree topologies defined by likelihood has not been shown to be NP-complete (Felsenstein, 2004), but remains extremely challenging because branch-and-bound search is not yet effective for trees represented in this way.
This exhaustive method tries to search all the possible tree topologies and considers every position in the alignment (not only the informative sites). Foster and Hickey (1999) applied this method.

The advantages of Maximum Likelihood are the following:

— has often lower variance than other methods (i.e. it is frequently the estimation method least affected by sampling error)
— tends to be robust to many violations of the assumptions in the evolutionary model
— even with very short sequences, it tends to outperform alternative methods such as parsimony or distance methods
— it uses all the sequence information

The disadvantages are:

— very CPU intensive and thus extremely slow
— the result is dependent of the model of evolution used

In the remaining of this work we will focus on Maximum Parsimony as it is an interesting optimization problem and the subject of this thesis.

2.3 Problem statement

Let $\mathscr{S} = \{S_1, S_2, \ldots, S_n\}$ be a set of n taxa [1] of length k that represent existing species and that are defined over an alphabet $\mathscr{A}$. In the case of DNA $\mathscr{A}$ is composed of four nucleotides, the gap symbol - and ? which represents a missing or unknown symbol: $\mathscr{A} = \{-, A, C, G, T, ?\}$. Generally, $\mathscr{S}$ is the result of a multiple alignment that in its turn relies on distance methods (see section 2.2) to define the order in which taxa are chosen (for example with Clustal). A *phylogenetic binary tree* $T = (V, E)$ (see Figure 2.3) which represents ancestral relationships between species is composed by a set of nodes $V = \{v_1, \ldots, v_r\}$ and a set of edges $E \subseteq V \times V = \{\{u, v\} | u, v \in V\}$. The set of nodes V of size $(2n - 1)$ is divided in two subsets:

1. *taxa* is the plural of taxon which stands for taxonomic unit, we can also use the term OTU which stands for Operational Taxonomic Unit.

— L is a subset of n *leaves*, i.e., nodes without descendants or OTUs,
— I represents the $n-1$ *internal nodes* (also called hypothetical ancestors or HTUs [2]), each one has two descendants.

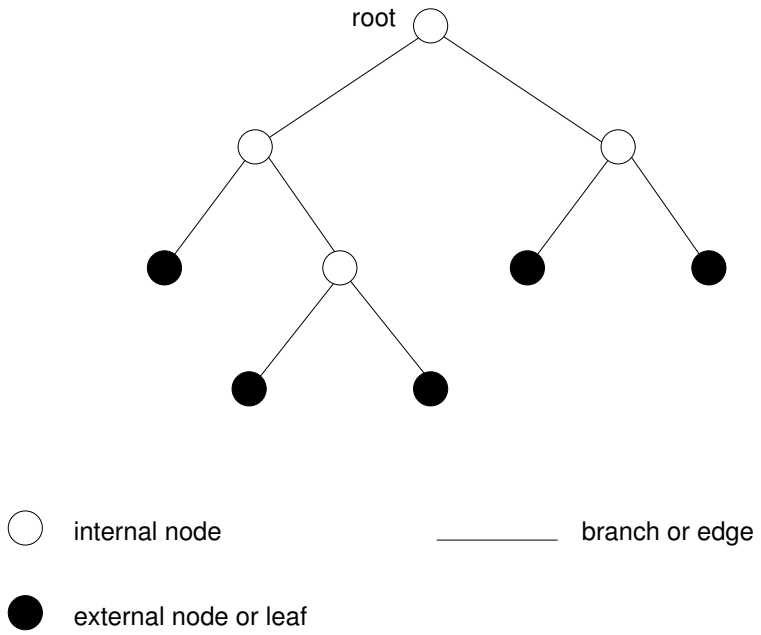
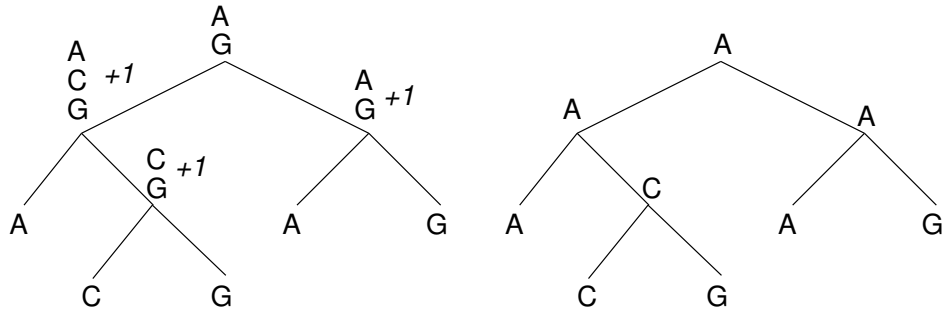


Figure 2.3 – Characteristics of a binary tree with 5 leaves and 4 internal nodes

There is generally a distinction made between what are called the small and large parsimony problems :

Definition 1 (Small parsimony problem) *Given a set $\mathscr{S}$ of n taxa of length k and a binary tree T whose leaves are labeled with taxa of $\mathscr{S}$, find the parsimony score of T.*

Note that the parsimony score is also called *tree length* by the cladistic community.



a) first−pass from leaves to root b) assignment after second−pass

Figure 2.4 – First pass and second pass for a tree of score of 3 under Fitch's optimality criterion

In order to compute the overall score of a tree, Fitch's algorithm (Fitch, 1971) starts from the root and goes down to the leaves and then gradually moves back from the leaves to the root in order to compute hypothetical ancestral taxa. This is often referred to as the *first-pass* of the algorithm. At each position of an internal node a set of bases is assigned. If two descendants x

2. *HTU*: Hypothetical Taxonomic Unit

and y of an internal node z have some bases in common they are assigned to the internal node $z_i = x_i \cap y_i$ for the *i-th* residue of the taxon. Otherwise all bases of both descendants are assigned to the parent $z_i = x_i \cup y_i$ and a cost of one unit is added to the overall score of the tree (see Figure 2.4) as it corresponds to a mutation. The *second-pass* of the algorithm, which starts from the root and reaches the leaves, enables to assign one nucleotide for a site if many possibilities exist, in order to obtain a hypothetical tree. Only the first-pass is necessary to obtain the parsimony score. More formally, the parsimony sequence z of every internal node $z = (x, y) \in I$ whose descendants are x and y and are represented by the sequences $S_x = \{x_1, \cdots, x_k\}$ and $S_y = \{y_1, \cdots, y_k\}$ is computed with the next expression:

$$\forall i, 1 \leq i \leq k, \quad z_i = \begin{cases} x_i \cup y_i, & \text{if } x_i \cap y_i = \emptyset \\ x_i \cap y_i, & \text{otherwise} \end{cases} \tag{2.3}$$

Consequently, the parsimony score of the sequence z (or number of mutations) is defined as:

$$\phi(z) = \sum_{i=1}^{k} C_i \qquad \text{where} \qquad C_i = \begin{cases} 1, & \text{if } x_i \cap y_i = \emptyset \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

and the parsimony score of the tree T is computed with the next equation:

$$\phi(T) = \sum_{\forall z \in I} \phi(z) \tag{2.5}$$

i.e. the sum of all mutations.

In contrast, the large Maximum Parsimony problem, which is far more complex, consists in finding a tree topology T^* where $\phi(T^*)$ is minimum, indeed it is a minimization problem:

Definition 2 (Large parsimony problem or Maximum Parsimony problem) *Given a set $\mathscr{S}$ of n sequences of length k find a most parsimonious tree T, i.e. a tree with minimum score.*

$$\phi(T^*) = \min\{\phi(T) : T \in \mathscr{T}\} \tag{2.6}$$

where $\mathscr{T}$ is the set of all possible tree topologies or search space.

In the following we will use the **Newick** notation to display trees which is a standard for representing trees in computer-readable format using parenthesis and commas. For example, the trees on Figure 2.5 will be represented respectively by : (A, B), $((A, B), C)$ and $(A, (B, C))$.

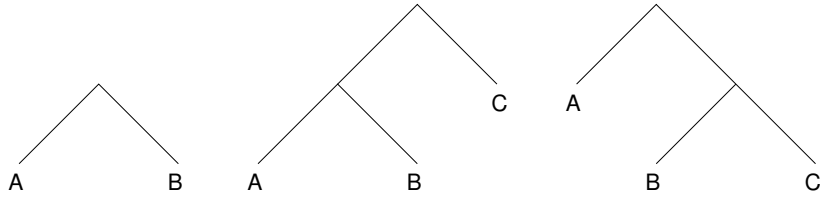


Figure 2.5 – Newick tree representation

As an example, we provide the graphics of figures 2.6, 2.7, 2.8. The initial tree as a parsimony score of 4 (green rectangle) and we put into red circles the local parsimony score of each internal node. The first move (degraph, regraph) (see Figure 2.7) enables to obtain a tree of score 3. The last move (see Figure 2.8) will generate a tree with an optimal parsimony score of 2.

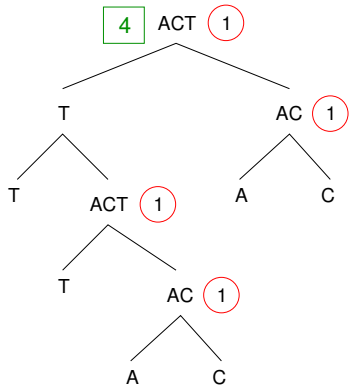


Figure 2.6 – Initial tree from which the right subtree (A,C) will be degraphed and regraphed on the left subtree (A,C)

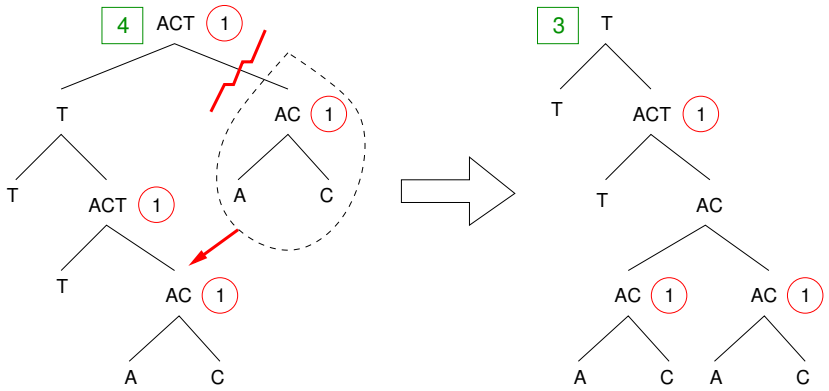


Figure 2.7 – Perform degraph of right subtree (A,C) and regraph on left subtree (A,C) to obtain a tree of score 3

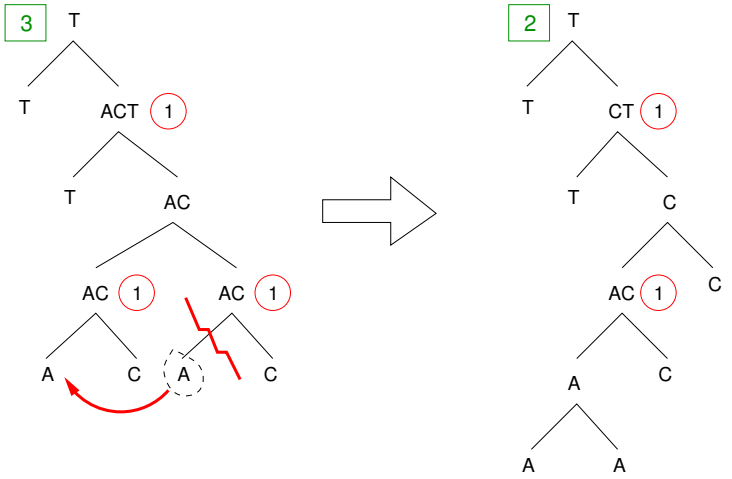


Figure 2.8 – Perform degraph of right leaf A and regraph on left leaf A to obtain a tree of score 2

2.4 Computational complexity

The cost of a tree can be computed in polynomial time (see (Fitch, 1971)). A rooted binary tree of n leaves has $n-1$ internal nodes, thus the complexity of the small parsimony problem is $O(n \times k)$. Indeed, we need to compute the hypothetical sequences of $n-1$ internal nodes. However, the search for an optimal tree is computationally intractable: the large parsimony problem is extremely difficult to solve since it is equivalent to the *NP-complete* Steiner problem in a hypercube (Foulds and Graham, 1982). This is why, as we shall see later on, heuristics methods constitute the main alternative in order to obtain near-optimal trees with reasonable computation time (Goloboff, 1993; Nixon, 1999).

2.4.1 Complexity analysis

The MP problem is NP-complete (Gusfield, 1997). This is easily proved because it is equivalent to the problem of Steiner tree in a hypercube, which is known in combinatorial optimization as a NP-complete problem (Garey and Johnson, 1977).

Definition 3 (Hypercube) *A hypercube d is a undirected graph with nodes 2^d bijectively tagged, i.e., each tag belongs to only one element in the graph and all elements in the graph belong at least to one element of the group of tags, where the tags are represented by the integer numbers between 0 and $2^d - 1$. Two nodes in the hypercube are adjacent if and only if, the binary equivalent of their tags differs by only one bit.*

In order to easily understand the reduction of the MP problem to the problem of Steiner tree in a hypercube, we will take as a basis the work of Goëffon (2006), where a phylogenetic tree could be seen as a set of points connected in a sequence inside a hypercube. We will consider two binary sequences S_1 and S_2, with $\mathscr{A} = \{0, 1\}$ and 3 points in simultaneous way, where every sequence is composed by one or two characters (see the table 2.1).

Sample 1	Sample 2	Sample 3
S_1 0	S_1 00	S_1 00
S_2 1	S_2 01	S_2 11

Table 2.1 – Three samples with two binary sequences S_1 and S_2

Now, imagine an hypercube of m dimensions, whose nodes are represented by each one of 2^m different binary sequences, which in accordance with the definition, are considered adjacent only if their Hamming distance is one.

It should be noticed that with two taxa, only one phylogenetic tree could be constructed. On Figure 2.9 we represent a tree in a hypercube, where the biggest circles correspond to the sequences of the problem and are tagged with each one of the $2^m - n$ possible sequences. In this context the MP problem consists in finding the shortest tree in the hypercube that at the same time contains all nodes corresponding to the specific data. Every edge in the tree represents a state change, and the parsimony score of the tree is equal to its number of edges (length).

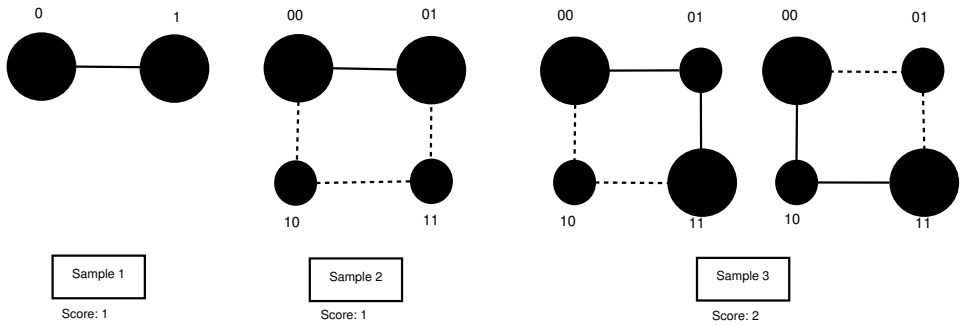


Figure 2.9 – Example of the MP problem representation of in a hypercube

The resulting tree is not always a binary tree, but it is easy to transform it. First the nodes of grade 3 are inserted where some nodes in the hypercube have a superior grade. Then, particular nodes are extracted from the hypercube to be transformed in leaves if their grade is 2. Finally, in order to complete the binary tree, it is sufficient to replace all the pathways $(x_i, \ldots, x_{i+l})$ where only x_i and x_{i+l} have a different grade of 2, by a simple edge (x_i, x_{i+l}) tagged with the length l of the pathway. In this way, a solution without root is obtained and it is possible to insert a root in any part in the tree.

In Figure 2.9, the sample 1 represents a hypercube of dimension one, which only have a solution of the problem. In the *sample 2*, almost of the trivial topology of a tree with two leaves, it is represented the shortest tree that optimizes the result of parsimony. In the same way in the *sample 3*, there are two possible explanations: an hypothetical ancestor tagged with 01 and another one with 10.

Definition 4 (Steiner tree.) *Let $G = (V, E)$ be an undirected graph. V is the set of nodes and $E \subseteq V^2$ are the edges. In each edge $(v_i, v_j) \in E$ a non negative weight is associated $\omega_{ij} \in \mathbb{N}^*$. We consider $X \subseteq V$ is a sub-set of nodes of V. A Steiner tree $ST_G(X) = (V', E')$ is a tree such that $X \subseteq V' \subseteq V$ and $E' \subseteq E$. The nodes $V' \setminus X$ are Steiner points. The weight $ST_G(X)$, that is denoted as: $\Omega(ST_G(X))$, is defined by:*

$$\Omega(ST_G(X)) = \sum_{\substack{i,j \\ (v_i,v_j)\in E'}} \omega_{ij} \tag{2.7}$$

The Steiner problem called in this way by its creator the Swiss mathematician Jakob Steiner (1796-1863), consists in finding the tree with the minimum length given G and X. The Steiner problem is generated when the edges of G do not have an associated value. In this case, $\omega_{ij} = 1$ for all i, j such that $(v_i, v_j) \in E'$.

When the MP problem is mapped to binary sequences it is equivalent to the problem of the Tree Steiner in a hypercube. A hypercube of dimension m has a register for each one of the 2^m string of m bits. Two sequences separated by a state change, are represented by two adjacent vertexes in the hypercube. It is easy to see how the Steiner tree is the shortest, and is equivalent to the most parsimonious tree, tolerating the possibility of a non binary tree. The number of edges in the Steiner tree is equal to the minimum parsimony score. Sankoff and Rousseau (1975) were the first to establish a link between the MP problem and the Steiner problem

Back to the example proposed by Goëffon (2006), we consider five sequences of length 4 :

Sequence 1	0000
Sequence 2	0011
Sequence 3	0101
Sequence 4	0110
Sequence 5	1001

In order to solve the Steiner problem, we will consider an hypercube $H = (V, E)$ of dimension 4 and a set of five nodes $X = \{0000, 0011, 0101, 0110, 1001\}$. In Figure 2.10.a H is represented in the five squares denoted by the dark circles of higher size, as well as a Steiner tree of length 6. This tree is showed in Figure 2.10.b, where every edge represents a state change. The beginning nodes are showed with dark black while the other nodes represent the Steiner points.

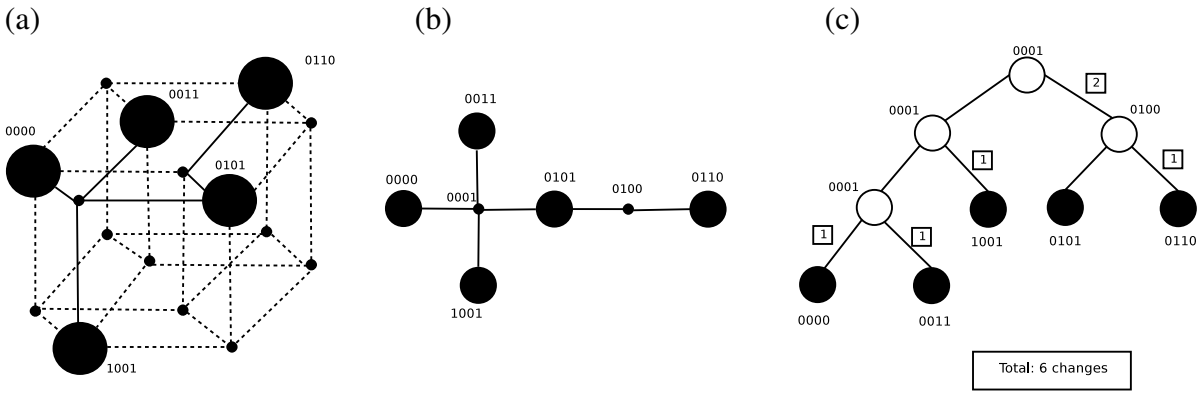


Figure 2.10 – Steiner problem in a hypercube dimension 4

In practice there is no need to go beyond, because the tree in Figure 2.10.b, by the elimination of the unnecessary Steiner points (level 2), shows all the necessary information. However, to achieve greater rigor and satisfy the Definitions 3 and 4, the tree is represented most clearly in Figure 2.10.c. We could find a score of 6.

In accordance with Foulds and Graham (1982), the Steiner problem in a hypercube is NP-complete, therefore the MP problem is NP-complete too.

2.4.2 Analysis of search space

The size of the search space for the MP problem is given by the number of the possible tree topologies for the set of n analyzed species. The next expression gives an estimation of the size (Xiong, 2006):

$$N_R = \frac{(2n-3)!}{2^{n-2}(n-2)!} \tag{2.8}$$

Where N_R is the total number of possible tree topologies with root. For example, if we apply the last equations for a set of

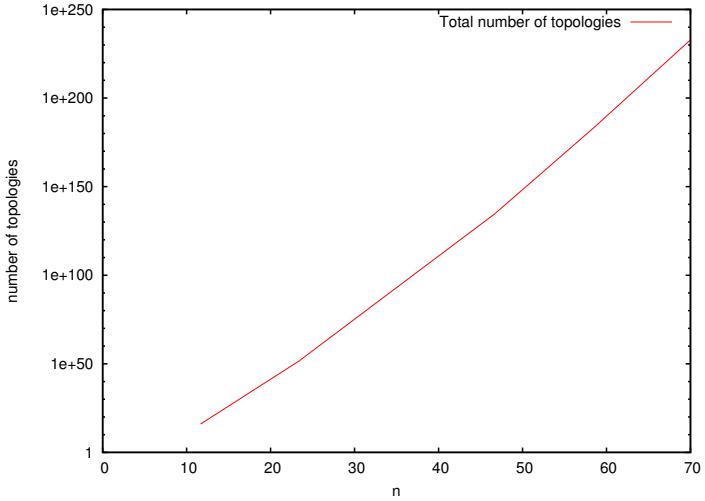


Figure 2.11 – Total number of tree topologies with root in function of n species.

$n = 6$ species the result would be 945 different topologies which is relatively small. But, for an instance with $n = 125$ species, handled commonly in the literature, the total number of possible rooted tree topologies is $N_R = 1.62129384\text{E}{+}243$.

In Figure 2.11 where we use a logarithmic scale we could see that the number of possible topologies of phylogenetic trees with root grows in function of the number n of species and the search space for this problem increases in factorial way in function of the value of n.

2.5 Resolution of MP

A thorough and historical introduction of methods to solve the Maximum Parsimony problem is available in Giribet (2007).

2.5.1 Exact methods

Exact methods explore the entire search space and provide the best solutions although they represent unfeasible methods to solve the problems due to the size of the search space which grows in an exponential way in function of the size of the instances. For the MP problem the size of an instance is given by the number of taxa studied n.

2.5.1.1 Exhaustive search

This method constructs all possible tree topologies then computes their parsimony score. Due to the prohibitive size of the search space (see Section 2.4.2), this method is impractical and may only be used to solve instances with less than ten taxa (Swofford, Olsen, Waddell, and Hillis, 1996a), from which all possible tree topologies are built.

2.5.1.2 Branch and bound algorithm

Branch and bound is a strategy to avoid exhaustive search. Theoretically, a branch and bound algorithm can not ensure polynomial time complexity in the worst case. Hendy and D. Penny (1982) were the first to implement a branch and bound algorithm which was designed to construct a minimum evolutionary tree of 11 species.

A first tree T_0 is generated (e.g. by a random process) and its score serves as an upper bound. Then a new tree T is built with 2 sequences. All remaining sequences are added iteratively on all possible branches creating new trees that are kept only if their score is lower than T_0. If by adding a taxon on a branch a tree T_i has a score greater than T_0 it is discarded.

Wu et al. (1999) used branch and bound to build trees with instances between 12 and 25 taxa. At first they build a $n \times n$ distance matrix M. Next the taxa are relabeled $(1, 2, \ldots, n)$ using a method to build an efficient permutation called MaxMin permutation. This method orders the taxa such that the taxon that has more characters in common with the others is in the first place and the last one is the taxon that has less characters in common.

Then they start to create the root v of the BBT (Branch and Bound Tree) such that v represents the only topology with leaves 1 and 2, and then the first tree is created using a modification of the algorithm UPGMA called UPGMM (Unweighted Pair Group Method with Maximum) and its score is stored in the variable UB. The building of the BBT then consists in adding one by one every taxon and if its score exceeds the value of UB all the nodes are deleted and they then build a new tree with the taxa that have not been used. They show that the efficiency of their algorithm with respect to the traditional method implemented by Hendy and D. Penny (1982) depends on the use of the MaxMin permutation.

Nevertheless, the fact that MP is a highly combinatorial problem makes exact methods impractical when the number of taxa exceeds 25 taxa for efficiency and storage reasons.

2.5.2 Heuristic methods

Heuristic methods try to generate optimal or near-optimal solutions to the problem in an iterative way. These methods are used when exact methods are unable to provide a solution in a reasonable amount of time. Although, the heuristic methods are generally more efficient than exact methods, they do not guarantee that generated solutions are optimal.

The first heuristic method of tree construction is the algorithm proposed by Wagner (1961) and implemented by Farris in 1970. Such trees were originally calculated by hand and used as final results to interpret a phylogeny. But it became evident that in the presence of homoplasy [3] *Wagner trees* were suboptimal solutions.

2.5.2.1 Greedy algorithms

Greedy algorithms are used in phylogenetic reconstruction under the name *stepwise addition* (Swofford, Olsen, Waddell, and Hillis, 1996b), *Random Addition Sequence* (RAS) (Goloboff, 1997) or *Wagner trees*. They use an optimization criterion: a taxon is inserted on a branch that minimizes the parsimony score of the resulting tree. It is possible that there are many positions with a minimum score; generally one is chosen randomly.

Andreatta and Ribeiro (2002) proposed a comparison between three greedy algorithms of different complexity and efficiency:

- *1stRotuGbr:* selects a taxon randomly at every iteration and inserts it in the position that minimizes the parsimony score of the tree.
- *Gstep_wR:* selects a taxon and a branch such that when the taxon is added in that position, the increase in the parsimony score is minimum.
- *GRstep:* is a variant of Gstep_wR where the selected branch and taxon do not increase the parsimony score more than ten

From their experiments they concluded that *Gstep_wR* is slightly more efficient than *1stRotuGbr* but its complexity ($O(n^3)$ vs $O(n^2)$) increases dramatically the runtime. *GRstep* is useful only when it is combined with local search.

Weber et al. (2005) explored the representation based on a greedy *decoder*, a parameterization of the greedy variant of stepwise addition. They consider that any permutation of taxa determines a tree, by execution of a greedy procedure, the permutation itself can be taken to be a representation of that tree. The score of a permutation is calculated by greedily decoding it into the corresponding tree and calculating the score of that tree. The random variant of the stepwise addition constructs a tree under the direct representation. Random trees under the greedy decoder representation can be generated by decoding randomly selected

3. *homoplasy*: correspondence between parts or organs acquired as the result of parallel evolution or convergence

permutation of the taxa. They compared both greedy and randomly built permutation to construct a greedy tree and they get a considerable difference between the two methods getting the best tree scores with the greedy decoder.

2.5.2.2 Stochastic local search (LS)

In *Combinatorial Optimization*, an instance of a problem consists of a set of feasible solutions and a cost function used to evaluate the quality of solutions. The cost function is also called *objective* function, *score* function or *fitness measure*. The goal is to find a solution with the optimal cost among all feasible solutions. Generally the problems addressed are computationally intractable, thus approximation algorithms have to be used. One class of approximation algorithms that have been surprisingly successful in spite of their simplicity are local search methods (Aarts and Lenstra, 2003; Papadimitriou and Steiglitz, 1998).

The MP problem has a huge search space; for that reason it was empirically proved that the methods of stochastic local search (or neighborhood search) are suitable to solve this problem (Ganapathy, Ramachandran, and Warnow, 2004). This kind of methods starts with an initial solution s_0, that is improved through an iterative search process. This process continues until a stop condition is satisfied, which could be the number of solutions examined.

Local search is based on the concept of neighborhood. A neighborhood N of a solution s is a set of solutions that are in some sense close to s, for example because they can be easily computed from s or because they share a significant amount of structure with s. We will say that s' is a neighbor of s by using the following notation: $s' \in N(s)$.

Definition 5 (Local optimum) *A local optimum of a combinatorial optimization problem is a solution that is optimal (either maximal or minimal) within a neighboring set of solutions. This is in contrast to a global optimum.*

Definition 6 (Global optimum) *A global optimum is the optimal solution among all possible solutions.*

In this sense the neighborhood has much influence on a search as it may, or may not, be able to generate the optimal solution.

2.5.2.3 Descent or branch-swapping.

The *descent* algorithm (also called *hill-climbing* in the case of maximization (see Algorithm 1) generates an initial solution s_0, then searches a close solution s' in the neighborhood of the current solution s, which should be better than s for the cost function. This process is repeated until there is no more improvement of s. In the case of MP, the descent is also called a *round of branch-swapping* by biologists from the Clasdistics community. A neighborhood function is then a *branch-swapping* for them.

Algorithm 1: Descent algorithm

```
1 Descent(s_0, 𝒩)
  input: s_0 : initial solution, 𝒩 : a neighborhood
  output: best solution found
2 s ← s_0
3 iteration ← 1
4 while ∃s' ∈ 𝒩(s), s.t.  fitness(s') < fitness(s) do
5     s ← s'
6     iteration ← iteration + 1
7 return s
```

Weber et al. (2005) made a comparison between different methods to construct phylogenetic trees using a permutation either random or greedy for the initial solution, then they compared their algorithm called Greedy High Climbing (GHC) using a new perturbation method that exchanges the positions of two pairs of leaves. They compared their results against the efficiency of

the TBR (see next Section) neighborhood function and showed that this new method of perturbation is not always more efficient than TBR.

The descent algorithms are the base of many methods that efficiently solve the MP problem.

2.5.2.4 Neighborhood functions.

We now come to the description of the main neighborhood functions found in the literature relative to trees:

Nearest neighbor interchange (NNI)

NNI is a technique proposed by Waterman and Smith (1978) which consists in exchanging two branches separated by an internal node. The size of the neighborhood is $O(n)$. Each tree has $2n - 6$ NNI neighbors, $n - 3$ internal branches and two possible movements by branch (Robinson, 1971). Figure 2.12 shows how NNI works. From an initial tree $((A, B), (C, D))$ we can generate two neighbors $((A, C), (B, D))$ and $((A, D), (B, C))$. NNI can be considered as a small size neighborhood.

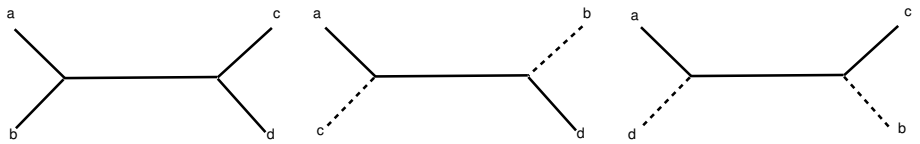


Figure 2.12 – Example of neighborhood NNI

Subtree pruning regrafting (SPR)

SPR (Swofford and Olsen, 1990) cuts an internal subtree t from a tree T and reinserts it elsewhere in the remaining subtree $T - t$. There are $2(n - 3)(2n - 7)$ possible SPR rearrangements (Allen and Steel, 2001) for each tree which makes it a medium size neighborhood $O(n^2)$. Figure 2.13 shows how this neighborhood works.

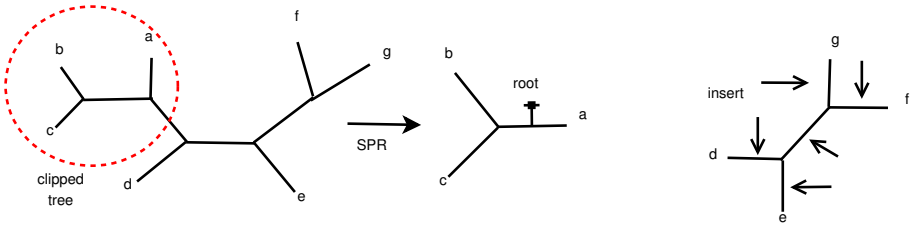


Figure 2.13 – Example of neighborhood SPR

Tree bisection reconnection (TBR)

TBR (Swofford and Olsen, 1990) consists in breaking the tree in two subtrees which will be reconnected from one of their branches. It can be considered as a *double* SPR where all possibilities of reconnection are tried: t is re-inserted on $T - t$ and $T - t$ is reinserted on t. From a given tree, the TBR neighborhood induces at most $(2n - 3)(n - 3)^2$ neighbor trees or $O(n^3)$ (Allen and Steel, 2001) and is a large size neighborhood. See Figure 2.14.

The following property holds for the neighborhoods: $NNI \subseteq SPR \subseteq TBR$. A more thorough study shows that NNI is too simple to generate some topologies and will let the descent algorithm get stuck quickly. TBR as a large size neighborhood can have more influence and will sometimes enable to find high quality solutions. SPR represents a trade-off between efficiency and quality.

Single step (STEP)

STEP (Swofford and Olsen, 1990; Andreatta and Ribeiro, 2002) takes out a taxon (i.e. a leaf) from the current solution and puts it back into another branch of the tree (see Figure 2.15). Since each taxon may be reinserted into $2(n - 1) - 3 - 1$ different branches; each solution has $2n(n - 3)$ neighbors.

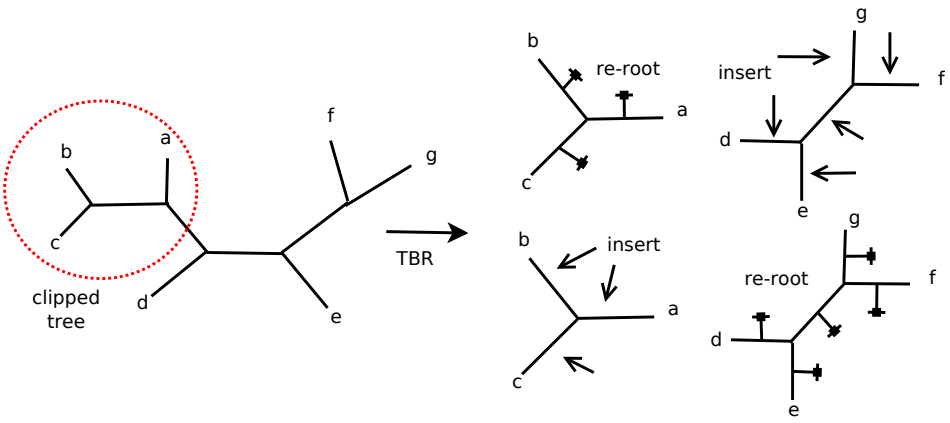


Figure 2.14 – Example of neighborhood TBR

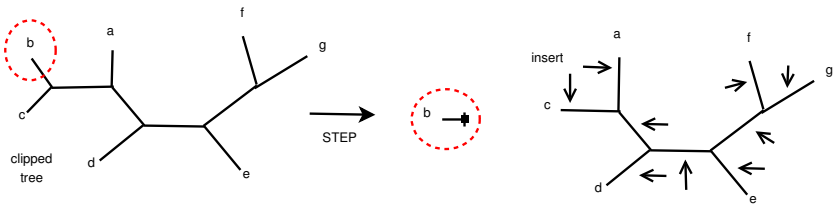


Figure 2.15 – Example of neighborhood STEP

2.5.2.5 Variable neighborhood search (VNS)

Variable Neighborhood Search (VNS) (Hansen, Mladenović, and Moreno Pérez, 2010) proposed by Mladenović and Hansen (1997) is a metaheuristic method to solve a set of combinatorial optimization and global optimization problems. It explores different neighborhoods during the descent. When the descent is stuck on a local optimum, it uses the next neighborhood. The core idea is to use small size neighborhoods at the beginning of the search to quickly reach a solution of good quality and then use large size neighborhoods to improve the final solution.

According to Mladenović and Hansen (1997), VNS is a metaheuristic which systematically performs the procedure of neighborhood change, both in descent to local minimum and in escape from the valleys which contain them.

The variable neighborhood descent algorithm proposed by Ribeiro and Vianna (2005) (see Algorithm 2) follows this model to solve the MP problem, it has a good behavior but its runtime is important.

Ganapathy et al. (2004) implemented this algorithm to solve the MP problem. They describe a new tree-rearrangement operation called p-ECR move, for p-Edge-Contract-and-Refine. Their algorithm computes the best 2-ECR neighbors of a given tree, based upon a simple data structure which also allows to calculate the best neighbor under NNI, SPR and TBR. They shows

Algorithm 2: Variable Neighborhood Search Algorithm

VNSDescent(s_0, $\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_l\}$)
input: s_0 : initial solution, $\{\mathcal{N}_1, \mathcal{N}_2, \ldots, \mathcal{N}_l\}$: set of neighborhoods
output: best solution found $s^\star$
$s \leftarrow s_0$
$s^\star \leftarrow s$
$i \leftarrow 1$
while $i \leqslant l$ **do**
  $s' \leftarrow$ **Descent**($s, \mathcal{N}_i$)
  if $fitness(s') < fitness(s^\star)$ **then**
    $s^\star \leftarrow s'$
  $s \leftarrow s'$
  $i \leftarrow i + 1$
return $s^\star$

that the use of 2-ECR in conjunction with TBR and/or NNI moves, may be a more effective technique for exploring tree space than TBR alone. The algorithm uses a preprocessing step, in which they assign three labels to each node in the tree in order to assign three optimal state-assignment labels to each internal node.

2.5.2.6 Progressive neighborhood

Based on the observation that important topological modifications of the tree are only performed at the beginning of the descent, Goëffon et al. (2008) have proposed a *Progressive Neighborhood* (PN), which contrary to VNS starts with a medium size neighborhood (SPR) and is iteratively reduced to NNI. Results show that PN needs a smaller number of iterations than traditional SPR searches to obtain solutions of the same quality by limiting the evaluation of relevant configurations. Recently PN has been used by Pirkwieser and Raidl (2008) to find consensus trees of high quality.

In order to evolve neighborhoods, a topological distance on trees is defined in (Goëffon, Richer, and Hao, 2008) that enables to build a distance matrix for a set of taxa given a tree topology. This distance is also used to control the size of the neighborhood (i.e. the distance between a pruned edge and its inserted edge is at most equal to a given limit).

Definition 7 (Topological distance) *Let i and j be two taxa of a tree T. The topological distance $\delta_T(i, j)$ between i and j is defined as the number of edges of the path between parents of i and j, minus 1 if the path contains the root of the tree.*

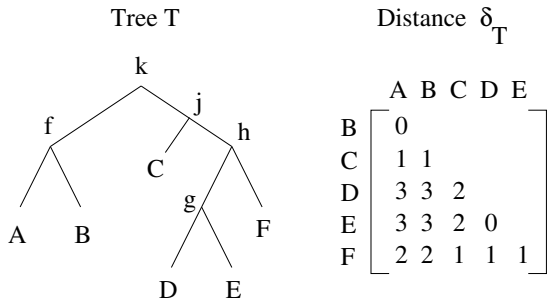


Figure 2.16 – Example of topological distance δ_T

For example, on Figure 2.16, A and B have the same parent f, so $\delta_T(A, B) = 0$, and $\delta_T(A, D) = 3$, because the number of edges between f and g is 4 ($f \leftrightarrow k \leftrightarrow j \leftrightarrow h \leftrightarrow g$), and as we pass through the root node k, we decrease the value by one unit. Note that for the topological distance, we consider trees as unrooted, this is why we remove one unit when passing through the root node. Progressive Neighborhood based on the topological distance was implemented in the software Hydra (Goëffon, Richer, and Hao, 2008). The process used in Hydra to reduce the size of the neighborhood takes into account a parameter M which corresponds to a maximum number of *Local Search* iterations. A parameter d is introduced to control the size of the

neighborhood and is defined as the maximal distance between a pruned edge and the edge where it is reinserted (i.e. distance δ between their two descendant nodes). As such, changing d leads to neighborhoods of different sizes which are explored with a descent algorithm.

2.5.2.7 Iterated local search (ILS)

There are different ways to obtain a new solution of a neighborhood when the descent algorithm is stuck in a local optimum. Some algorithms generate a new initial solution and explore new neighborhoods while other such as Iterated local search (ILS) (Charon and Hudry, 2002) disturb the optimum before to restart the descent (Lourenco, Martin, and Stützle, 2002).

The ILS metaheuristic iteratively constructs a sequence of solutions by embedding a local search. ILS has four essential elements: initial solution, local search, perturbation and acceptance criterion of the local search (see Algorithm 3).

Algorithm 3: Iterated Local Search algorithm for MP

```
1 IteratedLocalSearch(s_0, N, stop_condition)
  input: s_0 : initial solution, N : a neighborhood
  output: best solution found s*
2 s ← s_0
3 while not(stop_condition) do
4     s' ← Descent(s, N)
5     if fitness(s') < fitness(s*) then
6         s* ← s'
7     s ← perturbation(s')
8 return s*
```

ILS starts from a solution s_0 and performs a first descent until a local optimum s^* is found. Then it applies a perturbation method in order to generate an intermediate state. The next step consists in applying local search to the new solution. The algorithm iterates until a stop condition is satisfied.

Vázquez-Ortiz and Rodríguez-Tello (2011) applied the ILS algorithm to the MP problem and compared it against a simulated annealing (SA) algorithm. Their experiments showed that the SA provides best results than ILS.

2.5.2.8 Simulated annealing (SA)

Simulated Annealing is a metaheuristic based on the work of Metropolis et al. (1953) in the field of statistical thermodynamic, where the process of annealing is modeled by the simulation of the energy changes in a system of particles. The temperature decreases until it converges to a freezing state. Kirkpatrick et al. (1983) and Cerny (1985) worked independently and showed how this process could be applied to optimization problems, associating key concepts of the simulation process with elements of combinatorial optimization.

Algorithm 4 shows the basic method of simulated annealing for minimization problems with a *Markov Chain* to sample solutions iteratively and stochastically.

Parameter t represents the temperature. A solution with an increase Δf in the cost function will be accepted with probability $e^{(-\Delta f/t)}$.

In phylogenetic reconstruction this metaheuristic was applied by Barker (2003) in his software named LVB [4]. It starts with a tree of random topology improved by a descent using alternately NNI and SPR (Swofford and Olsen, 1990). The algorithm uses the temperature as an arbitrary control parameter varying between 0 and 1.

During the search, changes that do not improve the tree score are accepted with probability $p = e^{-(1/T)\Delta f}$, where T is the current temperature and Δf is the change (increase of the score).

Vázquez-Ortiz and Rodríguez-Tello (2011) have implemented a simulated annealing algorithm analyzing carefully the different possibilities for the key components in order to find the combination that could offer the best quality solution to the problem at a reasonable computational effort. They analyzed two initialization methods (greedy and random), different combinations of neighborhoods, cooling schedules and stop conditions. They compared the best combinations of parameters with the best results

4. *LVB*: Ludwig Van Beethoven, because the author was listening to Beethoven when he was implementing the software.

Algorithm 4: Simulated Annealing algorithm

SimulatedAnnealing(s_0, $\mathscr{N}$, T_0, T_f, α)
input: $\mathscr{N}$: neighborhood, f : fitness function, MCL : Markov Chain length, α : cooling scheme, T_i : initial temperature, T_f : final temperature
output: best solution found $s^\star$

$s_0 \leftarrow GenerateInitialSolution()$
$s \leftarrow s_0$
$s^* \leftarrow s_0$
$t \leftarrow T_i$
while $t > T_f$ **do**
  $i \leftarrow 0$
  while $i < MCL$ **do**
    $s' \leftarrow GenerateNeighbor(s, \mathscr{N})$
    $\Delta f \leftarrow f(s') - f(s)$
    generate a random $u \in [0, 1]$
    if $(\Delta f < 0)$ **or** $(e^{-\Delta f/t} > u)$ **then**
      $s \leftarrow s'$
    if $f(s') < f(s^*)$ **then**
      $s^\star \leftarrow s'$
    $i \leftarrow i + 1$
  $t \leftarrow \alpha t$
return $s^\star$

known in the literature for a set of instances. They could improve 2 best-known solutions and match the results of 13 instances over 20.

The most recent work to solve MP with the simulated annealing algorithm was SAMPARS [5] implemented by Richer (2013) as the combination of the work of Richer et al. (2009) and Vázquez-Ortiz and Rodríguez-Tello (2011). SAMPARS provides the best results reported for the set of instances used in this work.

Another so-called implementation of SA is called Tree-Drifting. It is described in (Goloboff, 2002) as a TBR descent able to accept suboptimal trees. The key component of the method is the function designed to determine the probability of acceptance, which is based on both the absolute step difference and a measure of character conflict: the *relative fit difference*, which is the ratio of steps gained and saved in all characters, between the two trees. However there is no algorithm given for this method in the literature. Tree-Drifting is embedded in the software *TNT* (Goloboff, Farris, and Nixon, 2008a), but used alone, it does not provide results of good quality.

2.5.2.9 Tabu search (TS)

Tabu search is a local search metaheuristic algorithm created by Glover and McMillan (1986) and formalized in (Glover, 1989) and (Glover and Laguna, 1990).

Tabu search uses local search or neighborhood functions to move from one potential solution s_0 to an improved solution s. To avoid being stuck in poor-scoring areas and explore regions of the search space that would be left unexplored by other local search, TS uses memory structures known as tabu list, a set of rules and banned solutions used to filter which solutions will be admitted to be explored by the search. A tabu list is a short-term set of the solutions that have been visited in the recent past (see Algorithm 5). the number of previous solutions to be stored).

Lin et al. (2007) designed TABUPARS, that seems to be the only work in the literature to solve the MP problem using TS. For their experiments they used a data set obtained from nuclear ribosomal DNA sequences (instances of size between 10 and 20, which is relatively small). In order to prevent the search process from revisiting a tree that has been visited recently, they use a tabu list which is composed of two arrays L and N. L contains the leaves of the tree ordered according to their position in the tree (from left to right) and N contains the internal nodes ordered according to their positions. As neighborhood function they use the method of leaf swapping (exchange two leaves) and profile change (exchange two nodes).

5. *SAMPARS*: Simulated Annealing for Maximum PARSimony.

Algorithm 5: Tabu Search Algorithm

TabuSearch(s_0, $\mathcal{N}$)
input: s_0 : initial solution, $\mathcal{N}$: neighborhood
output: best solution found $s^\star$
$s \leftarrow s_0$
$s^\star \leftarrow s$
$i \leftarrow 1$
$TabuList \leftarrow \emptyset$
while $not(stop\_condition)$ **do**
  $s' \leftarrow$ **Descent**$(s, \mathcal{N})$
  if $s' \notin TabuList$ **then**
    if $fitness(s') < fitness(s^\star)$ **then**
      $s^\star \leftarrow s$
    $s \leftarrow s'$
    $TabuList \leftarrow add(TabuList, s)$
return $s^\star$

2.5.2.10 Greedy randomized adaptive search procedure (GRASP)

This procedure is divided in two phases. During the first one; a greedy algorithm is applied where a taxon is added step by step in the position that reduces the increase of the parsimony score. The hybrid method of addition suggests an exchange of branches every time that a taxon is added to the tree, in order to do a correction that improves the parsimony score.

During the second phase, local search is applied (see Algorithm 6). Ribeiro and Vianna (2005) applied the GRASP+VND algorithm to solve the MP problem, using some instances taken from the literature and other randomly generated for their tests, proving that the new heuristic showed best performance in the solution quality with respect to the algorithms reported in the literature at that time.

Algorithm 6: GRASP Algorithm

GRASP($max\_iterations$, $\mathcal{N}$)
input: $max\_iterations$: integer, $\mathcal{N}$: neighborhood
output: best solution found $s^\star$
$s^\star \leftarrow \emptyset$ // $fitness(\emptyset) = +\infty$
$i \leftarrow 1$
while $i \leqslant max\_iterations$ **do**
  $s_0 \leftarrow greedy\_random\_generation()$
  $s' \leftarrow$ **Descent**$(s_0, \mathcal{N})$
  if $fitness(s') < fitness(s^\star)$ **then**
    $s^\star \leftarrow s'$
  $i \leftarrow i + 1$
return $s^\star$

2.5.2.11 Genetic and memetic algorithms

The concept of genetic algorithms (GA) was introduced by Holland (1975) and popularized by Goldberg (1989). Genetic algorithms belong to the class of evolutionary algorithms (EA), which generate solutions to optimization problems using techniques inspired by natural evolution, such as inheritance, mutation, selection, and crossover. They are modeled loosely on the principles of the evolution via natural selection. They rely on a population of individuals (candidate solutions) that undergo selection in the presence of variation operators such as mutation and recombination (crossover). A fitness function is used to evaluate individuals, and reproductive success varies with fitness (see Algorithm 7).

The evolution usually starts from a population of randomly generated individuals. In each iteration (generation), the fitness of every individual in the population is evaluated. The fittest individuals are stochastically selected from the current population, and each individual is modified (recombined and possibly randomly mutated) to form a new generation. The new generation

Algorithm 7: Genetic Algorithm

```
1  GeneticAlgorithm(P, 𝒩, stop_condition)
   input: P : initial population, 𝒩 : neighborhood
   output: best solution found s*
2  generation ← 1
3  while not(stop_condition) do
4      (father, mother) ← ParentSelection(P)
5      child ← Crossover(father, mother)
6      child ← Mutation(child)
7      Replace(child, P)
8      generation ← generation + 1
9  s* ← best of P
10 return s*
```

of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

According to Hoos and Stützle (2005) in most cases a genetic algorithm is not effective enough, because the crossover and mutation operators do not increase efficiency of the search.

For this reason GA have been hybridized with local search, in which a local search method replaces the mutation operator or is applied after it. This drives the search in different areas while guiding it with the crossover and selection mechanisms. This kind of hybridized methods are known as memetic algorithms (Moscato, 1999) or genetic local search algorithms (Ulder, Aarts, Bandelt, van Laarhoven, and Pesch, 1991) (see Algorithm 8).

Algorithm 8: Memetic Algorithm

```
1  MemeticAlgorithm(P, 𝒩, stop_condition)
   input: P : initial population, 𝒩 : neighborhood
   output: best solution found s*
2  generation ← 1
3  while not(stop_condition) do
4      (father, mother) ← ParentSelection(P)
5      child ← Crossover(father, mother)
6      child ← Mutation(child)
7      child ← LocalSearch(child)
8      Replace(child, P)
9      generation ← generation + 1
10 s* ← best of P
11 return s*
```

Memetic algorithms have had a few applications in phylogenetic reconstruction. To date only the works of Matsuda (1996) and Lewis (1998) have been registered.

The most recent application of this algorithm to solve the MP problem was performed by Richer et al. (2009), with the software Hydra which is based on an integration of an effective local search operator with a specific topological tree crossover operator.

The crossover operator is based on a topological distance defined as follows: given i and j two taxa of a Tree T, the topological distance $\delta T(i, j)$ is defined as the number of edges of the path between parents of i and j, minus 1 if the path contains the root of the tree. This topological distance tries to keep the properties of the parent trees. The local search improves the quality of each created offspring by a descent algorithm using a neighborhood called Progressive Neighborhood (PN). The initial population is generated with the greedy algorithm *1stRotuGbr*. For the crossover they use the Distance-Based Information Preservation tree crossover (DiBIPX) which takes into account the topological distance of parents trees, the key idea is to preserve the topological distance of parent trees. Experimentations on a number of real benchmark instances from TreeBase show that Hydra competes very well when compared to TNT. Hydra is able to find phylogenetic trees of better parsimony score with much fewer evaluations of candidate trees. Results show that Hydra improved the results of *TNT* on 6 instances and matched TNT on

5 instances and got worse in only one.

2.5.3 Parallel algorithms

A parallel algorithm or concurrent algorithm is an algorithm which can be executed a piece at a time on many different processing devices, and combined together again at the end to get the correct result (Blelloch and Maggs, 1996).

The cost or complexity of serial algorithms is estimated in terms of the space (memory) and time (processor cycles) that they take. Parallel algorithms need to optimize an additional resource: the communication between different processors. There are two ways parallel processors communicate, shared memory or message passing.

In order to solve the MP problem in efficient way, some authors Snell et al. (2000) and Du et al. (2005) considered the parallel algorithms.

— Blazewicz et al. (2011) proposed a parallel adaptive neighborhood search method. The algorithm achieves super-linear speedup and find solutions of good quality. Their main goal is to develop a strategy of cooperation of distributed computational nodes using an adaptive memory heuristic. In this paper, parallel adaptive memory programming (AMP) algorithms for parsimonious phylogenetic tree construction are proposed. The term adaptive memory programming was introduced by Glover and, it was originally related to tabu search (Glover, 1989; Glover and Laguna, 1990). Three versions of the parallel local search algorithms were created. Each one of the versions was combined with three neighborhood (i.e. NNI, SPR and TBR).The solution space is explored by the local search procedure with an adaptive memory mechanism. The algorithms try to find a local optimum using NNI. When found, such an optimum becomes a starting point for the local search procedure combined with the SPR. Next, the solution becomes a starting point for the local search procedure combined with TBR. The parallel algorithms have the master-slave structure. At the beginning of computation, the master process generates 30 starting solutions, which are then sent on the request to slave processes. When a local optimum has been found by a slave, it is sent to the master process, which collects the local optima that are used to create new starting solutions for slave process.
The motivation for such strategy is the hypothesis that local optima can contain fragments of the global optimum. They considered two stop criteria. The first was a value of the solution obtained while the second was based on a runtime limit. They compare their results with (Ribeiro and Vianna, 2005), (Viana, Gomes, Ferreira, and Meneses, 2009) and (Goëffon, Richer, and Hao, 2008) using the instances ANGI, CARP, ETHE, GOLO, GRIS, ROPA, SCHU, TENU. They found the best solution for: ANGI, CARP, ETHE, GRIS, SCHU, TENU (see Table 5.3 in chapter 5 for the description of those problems).

— White and Holland (2011) presented XMP a new program for finding exact MP trees that comes in both serial and parallel versions. The parallel version uses a work-stealing algorithm to scale to hundreds of CPUs on a distributed-memory multiprocessor with high efficiency. For their experiments they used instances real and synthetic from (Bader et al., 2006) and other real datasets. These instances have between 12 and 36 taxa and the length of taxa varies between 64 and 10536. This work is limited by the number of taxa which is relatively small.

2.5.4 Quartets concept

The meaning of this concept applied to phylogenetic trees is to break a big problem in different small problems and to solve every one independently. At the end we join all the partial solutions in order to get only one solution of the whole problem. In this method a phylogenetic tree is divided in quartets (topology on any four labels from the tree S) (see Figure 2.17). Then for each quartet we could notice 3 different topologies, where the topology that reduces the parsimony score is selected.

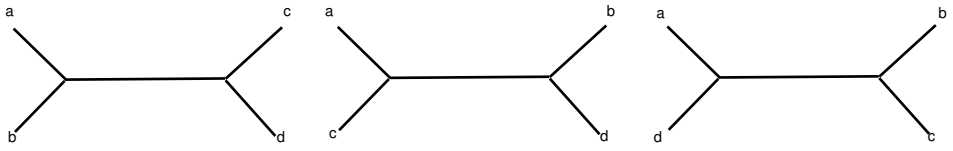


Figure 2.17 – The three possible quartets for the species set {a, b, c, d}.

The key idea is to consider small subsets of taxa, one at a time, and infer the phylogenies for these subsets. Given a complete set of correct quartets, a tree can be constructed in polynomial time.

Ben-Dor et al. (1998) use as input a list of weighted quartets over n taxa. Each quartet is a subtree on four taxa, and its weight represents a confidence level for the specific topology. The goal is to construct a binary tree with n leaves such that the total weight of the satisfied quartets is maximized. The first approach is based on geometric ideas, they embed the n points on the $n - dimensional$ unit sphere, while maximizing an objective function, which depends of the Euclidean distance between the four points and reflects the quartet topology. Given the embedding, a binary tree is constructed by performing geometric clustering, similar to neighbor joining, with the difference that the update phase retains geometric meaning: When two neighbors are joined together, their common ancestor is taken to be the center of mass of the original points. The second is based on dynamic programming, and it is guaranteed to find the optimal tree (with respect to the given quartets). They implemented both algorithms and ran them on real data for $n = 15$ taxa.

2.5.5 Reconfigurable computing

Reconfigurable computing is a computer architecture combining some of the flexibilities of software with the high performance of hardware by processing with very flexible high speed computing fabrics like field-programmable gate arrays (FPGAs). The principal difference when compared to ordinary microprocessors is the ability to make substantial changes to the data-path itself in addition to the control flow (Estrin, 2002).

Reconfigurable computing has been used to solve the MP problem by Kasap and Benkrid (2011) who used Maxwell, a FPGA based supercomputer developed by FPGA High Performance Computing Alliance (FHPCA) in Scotland. They use the Sankoff's algorithm to calculate the tree length. They take into account the levels of the tree; the first level is the closest to the leaves. The sub-trees in each level are computed in parallel until the root is reached. This is the first FPGA implementation but it supports a maximum of 12 taxa.

2.5.6 Solving MP as a multi-objective problem

New ideas have emerged in order to solve the MP problem under a multi-objective concept where one tries to optimize both the Maximum Parsimony and Maximum Likelihood versions of the problem.

As explained in the Section 2.2 different criteria have been employed to evaluate possible solutions to get the best phylogenetic tree in order to guide a search algorithm towards the best tree. However, these criteria may lead to distinct phylogenies, which are often conflicting among them. In this context Cancino and Delbem (2007) proposed a multi-objective evolutionary algorithm, named phyloMOEA, which employs the maximum parsimony and likelihood criteria to evaluate solutions. They consider that this approach can be useful since it could produce a spectrum of equally optimal trees (Pareto front) according to all criteria. They tested phyloMOEA using 4 datasets. The algorithm finds for all datasets a Pareto front representing a trade-off between the criteria. The main objective of phyloMOEA is the computation of a set of non-dominated solutions (trees), which represents a trade-off between parsimony and likelihood scores.

phyloMOEA is a multi-objective evolutionary algorithm that uses the following parameters:

— Initial population: creates random trees
— Fitness Evaluation: the parsimony and likelihood scores are calculated using Fitch and Felsenstein algorithms, respectively. The rank is calculated using a non-dominated sorting algorithm applied to a population for all generations
— Selection: tournament selection, picks two individuals at random and choose the best one
— Crossover operator: prune a subtree s_T from parent $T1$ and remove all leaves of s_T from $T2$. The offspring $T'1$ results from a regraph of s_T in $T2$
— Mutation Operator: NNI

At each iteration i, the algorithm divides $R = P_i \cup Q_i$ (where P is an initial population and Q the current population)

into several frontiers, denoted by $F_1, F_2, ..., F_j$. The first frontier (F_1) is formed by non-dominated solutions from R. Thus the second frontier F_2 is by non-dominated solutions from $R - F_1$. This process is repeated to $R - F_1 - F_2$, and so on, until R is empty. At the end of a PhyloMOEA execution, duplicate trees are removed from the final population. Finally, the Pareto optimal solutions are calculated.

Unfortunately phyloMOEA uses only a set of 4 instances of different size including the instance called zilla (500 taxa of 759 residues).

2.5.7 Other techniques

Some other techniques have been developed to tackle the problem of Maximum Parsimony. They were integrated in the sofware *TNT* (Goloboff, Farris, and Nixon, 2008a).

— Ratchet (Nixon, 1999): it consists in temporarily modifying the problem by selecting some sites or by changing the weight of some sites. Consequently the objective function is modified and it helps escaping from a local optimum,
— Tree fusing (TF) (Goloboff, 2002): given many trees of same score, this method uses sub-groups with an identical taxa composition but different topologies and switch them, the resulting trees are then evaluated; it can be considered as a kind of crossover operator,
— Sectorial search (SS) (Goloboff, 2002): consists in optimizing a subtree t of the tree T, in other words to find a topology of the subtree that can give the whole tree a lower score.

2.5.8 Software

Table 2.2 presents a list of software related to the resolution of Maximum Parsimony. We did not put in this list old software like *NONA* (Goloboff, 1997) or platforms like *MEGA*. A complete list can be found on the web page of Joe Felsenstein [6]. *POY* (Wheeler, Lucaroni, Hong, Crowley, and Varón, 2015) for example is close to *TNT*.

6. *Parsimony software list*: http://evolution.genetics.washington.edu/phylip/software.html#Parsimony

Name of Software	Author	Year	Metaheuristic
PHYLIP	J. Felsenstein	1980	greedy + descent with NNI
POY	W. Wheeler	1997	RAS + TBR
PAUP Phylogenetic Analysis Using Parsimony	D. L. Swofford	2003	many new algorithms
LVB (Ludwig Van Beethoven)	D. Barker	2003	Simulated Annealing
GRASP+VND	Ribeiro and Vianna	2005	GRASP
TABUPARS	Lin et al.	2007	Tabu Search
TNT	P. Goloboff	2008	RAS + SPR/TBR, SS, Ratchet, TF, TD
Hydra	Goëffon, Richer, Hao	2009	Memetic Algorithm
XMP	White and Holland	2011	Branch and Bound
SAMPARS	Vazquez-Ortiz, Richer, Lesaint, Rodriguez-Tello	2013	Simulated Annealing

Table 2.2 – Non exhaustive list of parsimony software

2.6 Conclusion

Maximum Parsimony is a challenging problem and much work has been devoted to its resolution that range from a simple descent to memetic algorithms. However, it is sometimes difficult to compare the different implementations as there is no standard benchmark like *BaliBase* (Nuin, Wang, and Tillier, 2006) for multiple sequence alignments methods. In Chapter 5 we describe a database of more than 12,000 problems that we have gathered in order to predict the MP score of a problem and that could be used to assess the performance of the search and the quality of the solutions obtained by parsimony software.

3

Simulated annealing for the *Maximum Parsimony* problem

3.1 Introduction

Simulated Annealing (SA) is a general-purpose stochastic optimization technique that has proved to be an effective tool for the approximation of global optimal solutions to many NP-hard optimization problems. However, it is well known that the design of an effective SA algorithm requires a careful implementation of some essential components and an appropriate tuning of the parameters used (Johnson, Aragon, McGeoch, and Schevon, 1989, 1991).

In this chapter we present an improved implementation of a SA algorithm (see Algorithm 4), that we called SAMPARS to find tree topologies (phylogenies) with near-optimal parsimony costs under Fitch optimality criterion. The algorithm is a simple cooling schedule with a Markov Chain process.

3.2 Simulated annealing

The simulated annealing algorithm is based on the analogy between the simulation of the annealing of solids and the problem of solving large combinatorial optimization problems (Kirkpatrick et al., 1983; Cerny, 1985).

In the field of statistical thermodynamic, annealing denotes a physical process in which the temperature of a solid is increasing to a maximum value at which all particles of the solid randomly arrange themselves in the liquid phase, followed by a cooling process where the temperature is decreasing slowly. During the process from hot to cold the particles are modeled to reach an optimal form (Metropolis et al., 1953).

Kirkpatrick et al. (1983) and Cerny (1985) worked independently and they showed how this process could be applied to optimization problems, associating key concepts of the simulation process with elements of combinatorial optimization.

In order to implement an optimized version of simulated annealing algorithm to solve the phylogenetic reconstruction (see Algorithm 9), we have identified the principal components of this algorithm and we have selected different values to find the combinations that could offer the best quality solutions with reasonable computational effort. We call this algorithm and the resulting software SAMPARS for **S**imulated **A**lgorithm for the **M**aximum **PARS**imony problem.

In the next section we will present all the implementation details of the SAMPARS algorithm, each of its components and the possible combinations of parameters to obtain the best quality solution.

Algorithm 9: SAMPARS algorithm

SimulatedAnnealing(s_0, $\mathscr{N}$, T_0, T_f, α)
input: $\mathscr{N}$: neighborhood, f : fitness function, MCL : Markov Chain length, α : cooling scheme, T_i : initial temperature, T_f : final temperature
output: best solution found $s^\star$

$s_0 \leftarrow GenerateInitialSolution()$
$s \leftarrow s_0$
$s^* \leftarrow s_0$
$t \leftarrow T_i$
while $t > T_f$ **do**
  $i \leftarrow 0$
  while $i < MCL$ **do**
    $s' \leftarrow GenerateNeighbor(s, i, \mathscr{N})$
    $\Delta f \leftarrow f(s') - f(s)$
    generate a random $u \in [0, 1]$
    if $(\Delta f < 0)$ **or** $(e^{-\Delta f/t} > u)$ **then**
      $s \leftarrow s'$
    if $f(s') < f(s^*)$ **then**
      $s^\star \leftarrow s'$
    $i \leftarrow i + 1$
  $t \leftarrow \alpha t$
return $s^\star$

3.2.1 Initial solution

The initial solution s_0 is the starting phylogenetic tree used for the algorithm to begin the search for better configurations in the search space $\mathscr{T}$. SAMPARS can create the starting solution using a *random* or *greedy* procedure. A greedy initial solution is supposed to guarantee a better quality for the final solution. Other initialization methods could be used:

— *upgma, nj*: distance methods like UPGMA (Sneath and Sokal, 1973) or Neighbor-Joining (NJ) (Saitou and Nei, 1987; Gascuel, 2000) as they have a low complexity
— *nj + fitch*: use of the NJ distance method where the distance is Fitch evaluation ϕ; at each step of the NJ algorithm we recompute the distance matrix using Fitch scoring function
— *nj + greedy*: we have also designed a NJ and greedy procedure which uses a random selection of half of the sequences to generate a first NJ tree, and adds the second half to this tree in a greedy manner

3.2.1.1 Influence of initial solution

We have tried to determine which initialization procedure was the most useful. The initial solution can sometimes influence the search. For example, with a descent algorithm (local search) that employs a NNI (*Nearest Neighbor Interchange*) neighborhood, if one starts from a solution with a high parsimony score then the final solution will generally be of poor quality as the algorithm will converge prematurely to a local optimum (Goëffon et al., 2008).

Tests performed on *set1* (see Table 5.3 in chapter 5 and Table 3.1) show that on average $random > upgma > nj > nj + greedy > nj + fitch > greedy$. In other words, the random initialization procedure produces initial solutions of worst quality while a greedy algorithm gives the best configurations. For *set2* (results not reported here), we get slightly different results: $random > upgma > nj > greedy > nj > greedy > nj + fitch$.

Table 3.1 – Score of initial solution for different initialization methods for set 1 of real instances

problem	*random*	*upgma*	*nj*	*greedy*	*nj + greedy*	*nj + fitch*	*best*
ANGI	425.80	240.00	234.00	229.42	228.82	226.00	226.00
CARP	1546.92	628.00	592.00	591.18	591.68	597.00	591.18
ETHE	898.58	399.00	385.00	388.84	386.32	382.00	382.00
GOLO	903.82	567.00	535.00	536.32	534.16	533.00	533.00
GRIS	476.50	200.00	195.00	182.22	184.48	186.00	182.22
ROPA	731.28	363.00	353.00	345.72	346.00	347.00	345.72
SCHU	2627.02	880.00	842.00	803.22	812.64	805.00	803.22
TENU	1604.18	748.00	728.00	710.40	712.00	719.00	710.40

3.2.2 Neighborhood functions

The most common practice in the reported metaheuristics for the MP problem (Andreatta and Ribeiro, 2002; Ribeiro and Vianna, 2005, 2009) is to employ one of the three classical neighborhood functions: NNI, SPR, TBR (see Figure 3.1).

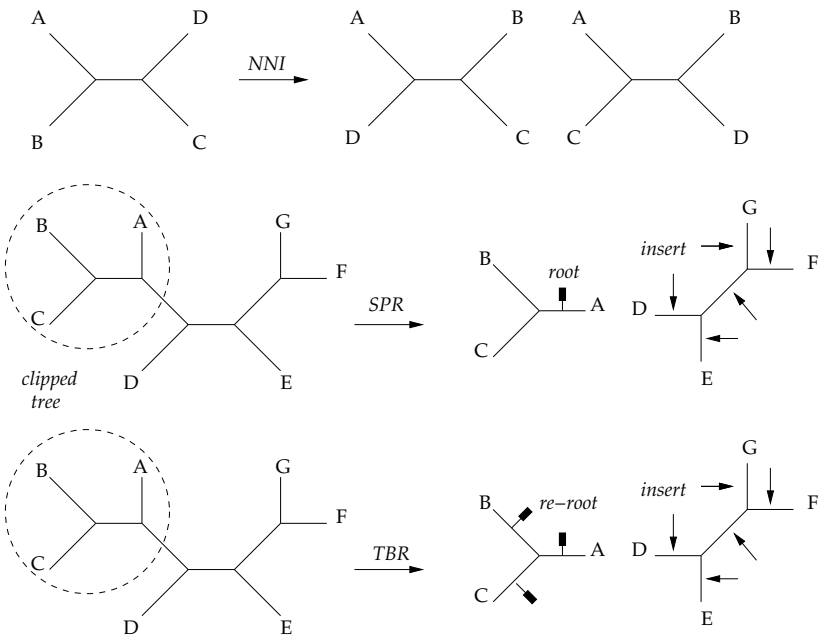


Figure 3.1 – NNI, SPR and TBR neighborhoods

LVB, an existing SA implementation for the MP problem (Barker, 2003, 2012) alternates the use of the NNI and SPR neighborhood functions at each iteration of the search process. In SAMPARS both the SPR and the TBR neighborhood relations are implemented but can only be used one at a time. However, from our preliminary experiments it has been observed that the separate use of these neighborhood functions is not sufficient to reach the best-known solutions, because both of them are highly disruptive. In order to achieve a better performance for SAMPARS, we have decided to use a third complementary neighborhood structure. It is based on a stochastic descent algorithm with a best-improvement scheme (see Algorithm 10) which is occasionally applied to the neighboring solution s' prior to returning it. Our neighborhood function is inspired by the ideas reported in (Lü et al., 2011), where the advantage of using this approach is well documented. The *BestImproveDescent* consists in a descent for which all neighbors of the current solution are processed and only the one that gives the best result is kept.

In Algorithm 10, every 25 iterations of the Markov Chain we improve the new configuration with a descent algorithm. This mechanism seems to greatly improve the parsimony scores. However, the frequency used (here 25) ideally should not be a constant but rather a proportion of the iterations per Markov chain (e.g. 1 out of 4 iterations) since the parameterization of SA is usually dependent on each problem instance it seems more sensible to use a ratio for frequency rather than an absolute value. But in order to to study the influence of the frequency of improvements we have decided to keep it a fixed value that we make

Algorithm 10: GenerateNeighbor function

input: s : current solution, $iteration$: iteration of Markov Chain, $\mathcal{N}$: neighborhood
output: best solution in the neighborhood s'

```
randomly select s' ∈ N(s)
if iteration is a multiple of 25 then
    s' ← BestImproveDescent(s)

return s'
```

Table 3.2 – Parsimony score for instance tst12 with nj+greedy for $freq = 5, \ldots, 70$

frequency	5	10	20	30	40	50	60	70
time (min)	72	31	17	13	10	9	8	8
average	1227.70	1226.50	1219	1219.90	1218.70	1218.50	1216.10	1214
minimum	1222	1222	1213	1215	1213	1210	1211	1208

range from 5 iterations or within 10 to 100 iterations (by step of 10) or 200 to 600 iterations (by step of 100).

3.2.2.1 Experiments and results obtained for the neighborhood function

Results obtained for different instances (see Tables 3.2 and 3.3 for instance tst12 in particular) show that:

— the computation time increases with a small frequency ($freq$) as there are more descents to perform: for example for instance tst12, with a frequency of 5, the computation time is around 72 minutes, while with a frequency of 10 it is only 31 minutes,
— if the frequency is too low (< 20) then it is more difficult to reach the best known optimal solution as we get stuck in local optima. For $freq = 10$, the average score of the solutions is 1226.50 and the minimum score found is 1222 while for $freq = 20$ the average score is 1219 and the minimum score found reaches 1213,
— if the frequency is too high (> 200) then it is more difficult to reach a good solution,
— depending on the problem, the interval for which we can obtain good results is $20 \leq freq \leq 100$.

3.2.3 Cooling schedule

A cooling schedule is defined by the following parameters: an initial temperature T_i, a final temperature T_f or a stopping criterion, the number of solutions generated at each temperature (Markov Chain Length $= MCL$), and a rule for decrementing the temperature. The cooling schedule governs the convergence of the SA algorithm. At the beginning of the search, when the temperature has a high value, the probability of accepting solutions of worse quality than the current solution (uphill moves) is high. It allows the algorithm to escape from local minimum. The probability to accept such moves is gradually decreased as the temperature decreases to zero.

The literature offers a number of different cooling schedules, see for instance (Aarts and Van Laarhoven, 1985; Van Laarhoven and Aarts, 1988; Abramson et al., 1999; Rodriguez-Tello et al., 2008). They can be divided into two main categories: static and

Table 3.3 – Parsimony score for instance tst12 with nj+greedy for $freq = 80, \ldots, 600$

frequency	80	90	100	200	300	400	500	600
time (min)	8	8	7	6	6	5	5	4
average	1217.40	1218,30	1218.40	1219,30	1219,80	1217,40	1220,30	1220,10
minimum	1215	1216	1215	1212	1216	1213	1213	1215

dynamic. In a static cooling schedule, the parameters are fixed and cannot be changed during the execution of the algorithm. With a dynamic cooling schedule the parameters are adaptively changed during the execution.

In the SAMPARS implementation we preferred a geometrical cooling scheme mainly for its simplicity, with a reheating system that makes it dynamic. It starts at an initial temperature T_i that can either be defined by the user or automatically computed using the following formula: $\sqrt[3]{(k+n)}$ which generates values under 6.0 for most of the tested benchmark instances, which was found to give good results. Then, this temperature is decremented at each round by a factor $\alpha = 0.99$ using the relation $t = \alpha t$. A reheat mechanism has also been implemented. If the best-so-far solution is not improved during 40 consecutive temperature decrements, the current temperature t is increased by a factor $\beta = 1.4$ using the function $t = \beta t$. In our implementation this reheat mechanism can be applied at most $max\_reheat = 4$ times, since it represents a good trade-off between efficiency and quality of solution found.

For each temperature t, the maximum number of visited neighboring solutions is MCL. It depends directly on the parameters n and k of the studied instance, since we have observed that more moves are required for bigger trees (Vázquez-Ortiz and Rodríguez-Tello, 2011). The three different values that MCL can take and that we call *effort* were empirically decided, are:

— small: $MCL = 15 \times (n + k)$
— medium: $MCL = 23 \times (n + k)$
— large: $MCL = 40 \times (n + k)$

The parameter values presented in this section were chosen based on the parsimony score in a preliminary experimentation (Vázquez-Ortiz and Rodríguez-Tello, 2011).

3.2.3.1 Influence of the reheat mechanism and effort level

In order to determine the influence of the reheat mechanism we ran a total of 26,000 tests for medium and large effort levels, different initialization methods ($random$, nj, $greedy$, $nj + greedy$, $nj + fitch$), with different improvement frequencies (20, 30, 60, 90) and reheat values that range from 0 (no reheat) to 4 (reheat is performed at most 4 times) and a SPR neighborhood.

Results are shown in Table 3.4. For each problem we provide the average parsimony score obtained, the minimum score that could be reached and the number of times it was reached out of 20 executions.

We can see that problem *tst10* does not seem to be too complex because we could find the minimum score of 720 for every configuration. However, we can see that permitting up to 4 reheats enables the minimum score to be found 38 times over all tests performed, i.e. a total of 400 (5 methods, 4 frequencies and 20 executions). Problem *tst20* is a more complex problem as the minimum score of 659 could only be found with a large effort level with a reheat of 1 or 4.

The results confirm our former observations: a large effort level will lead to results of better quality compared to a medium effort level and a reheat of 1 or 4 can provide better results.

In a second experimentation (Table 3.5) we have tested some methods to get the initial solution combined with effort level and a given number of reheats in order to assess if the initial solution has some influence on the final result. The answer, obtained from the results, is that the nj and $nj + greedy$ methods will on average provide better results and help to reach a maximum number of good solutions (see results in boldface).

3.3 Computational experiments

This section reports on three main experiments were conducted to evaluate the performance of the SAMPARS algorithm. The objective of the first experiment is to determine both a component combination, and a set of parameter values which enables SAMPARS to attain the best trade-off between solution quality and computational effort. The purpose of the second experiment is to carry out a performance comparison of SAMPARS with respect to an existing SA algorithm called LVB (Barker, 2003, 2012). The third experiment is devoted to assess the performance of SAMPARS with respect to three representative state-of-the-art procedures: GA+PR+LS (Ribeiro and Vianna, 2009), TNT (Goloboff et al., 2008a) and Hydra (Goëffon, 2006).

For these experiments SAMPARS was coded in C++ and compiled with *g++* using the optimization flag -$O3$. It was run on a cluster composed of Xeon X5650 CPUs under Linux 64 bits operating system. Due to the non-deterministic nature of the studied

Table 3.4 – Performance comparison for the reheat mechanism for problems *tst10* to *tst20*

effort level		medium					large				
reheat		0	1	2	3	4	0	1	2	3	4
tst10	Avg	723.12	722.67	722.70	722.81	722.52	722.33	722.10	722.05	721.98	721.99
	Best	720	720	720	720	720	720	720	720	720	720
	Times	12	23	14	10	16	23	27	36	32	38
tst11	Avg	544.73	544.20	544.09	543.98	544.00	543.96	543.62	543.58	543.41	543.42
	Best	541	541	541	541	540	541	540	541	540	540
	Times	5	5	4	5	1	11	2	6	2	1
tst12	Avg	1219.29	1218.91	1218.86	1218.59	1218.57	1217.37	1216.91	1217.21	1216.95	1216.92
	Best	1208	1208	1209	1208	1208	1208	1208	1208	1208	1208
	Times	1	1	2	1	1	1	4	3	3	2
tst13	Avg	1522.27	1521.94	1522.00	1522.00	1521.88	1521.11	1520.38	1520.47	1520.50	1520.47
	Best	1515	1515	1515	1515	1515	1515	1515	1515	1515	1515
	Times	2	4	6	5	4	8	17	11	12	12
tst14	Avg	1167.48	1166.90	1166.53	1166.74	1166.58	1166.15	1165.58	1165.48	1165.37	1165.33
	Best	1161	1160	1160	1161	1160	1160	1160	1160	1160	1160
	Times	2	2	2	1	1	4	2	2	4	5
tst15	Avg	757.40	757.03	756.95	757.04	756.90	756.12	755.99	755.81	755.62	755.75
	Best	752	752	752	752	752	752	752	752	752	752
	Times	6	8	6	6	6	6	8	13	16	17
tst16	Avg	534.50	534.17	533.84	533.87	533.99	533.48	532.98	532.88	532.96	532.88
	Best	528	527	528	529	528	527	527	527	528	527
	Times	3	3	1	3	3	2	6	3	3	3
tst17	Avg	2460.27	2459.39	2459.25	2459.14	2459.20	2457.76	2457.35	2457.21	2456.91	2456.85
	Best	2450	2450	2450	2450	2450	2450	2450	2450	2450	2450
	Times	3	2	1	4	4	3	9	4	6	13
tst18	Avg	1530.29	1529.59	1529.38	1529.42	1529.69	1527.57	1527.39	1526.88	1527.32	1526.97
	Best	1521	1521	1521	1521	1521	1521	1521	1521	1521	1521
	Times	10	18	22	22	16	31	36	46	39	43
tst19	Avg	1020.82	1020.27	1020.01	1020.00	1020.01	1019.19	1018.69	1018.77	1018.42	1018.61
	Best	1013	1013	1012	1012	1013	1012	1012	1012	1012	1012
	Times	2	1	1	1	1	1	1	1	4	2
tst20	Avg	666.81	666.33	666.33	666.35	666.21	665.68	665.28	665.18	665.22	665.28
	Best	660	662	661	660	661	660	659	661	661	659
	Times	1	6	2	1	4	1	1	9	4	1

Table 3.5 – Performance comparison for the initialization methods for tst10 to tst20

effort level	medium					large					
max reheats	0	1	2	3	4	0	1	2	3	4	total
$random$	0	1	0	0	0	3	4	2	2	4	43
$greedy$	0	1	0	0	1	1	4	1	4	3	47
$nj + fitch$	0	1	0	1	1	0	3	2	1	4	48
nj	1	1	2	0	0	2	3	3	2	**6**	63
$nj + greedy$	0	2	0	2	0	2	3	**5**	**5**	3	66
$allmethods$	0	0	0	0	0	0	5	1	2	5	156

algorithms, 30 independent runs were executed for each of the selected benchmark instances in each experiment presented in this section.

3.3.1 Benchmark instances and performance assessment

The test-suites that we have used in our experiments are the same proposed by Ribeiro and Vianna (Ribeiro and Vianna, 2003, 2005) and later employed in other works (Goëffon, 2006; Ribeiro and Vianna, 2009). It consists of two sets (see section 5.2). A first set (*set1*) of 8 benchmarks obtained from real data. A second set (*set2*) of 20 randomly generated instances (tst01 to tst20) with a number of sequences (n) ranging from 45 to 75 whose length (k) varies from 61 to 159. The *set2* is composed of some problems for which it is hard to reach the global optimum while for *set1* the instances are quite easy to solve. The criteria used for evaluating the performance of the algorithms are the same as those used in the literature: the best parsimony cost found for each instance (smaller values are better) and the CPU time in seconds.

3.3.2 Components and parameters tunning

Optimizing parameter settings is an important task in the context of algorithm design. Different procedures have been proposed in the literature to find the most suitable combination of parameter values (de Landgraaf et al., 2007; Gunawan et al., 2011). In this work we employ a tuning methodology based on *Combinatorial Interaction Testing (CIT)* (Cohen et al., 1996). We have decided to use *CIT*, because it allows to significantly reduce the number of tests (experiments) needed to determine the best parameter settings of an algorithm. Instead of exhaustively testing all the parameter value combinations of the algorithm, it only analyzes the interactions of t (or fewer) input parameters by creating interaction test-suites that include at least once all the t-way combinations between these parameters and their values.

Covering arrays (CAs) are combinatorial designs which are extensively used to represent those interaction test-suites. A covering array CA$(N; t, k, v)$, of size N, strength t, degree k, and order v, is an $N \times k$ array on v symbols, such that, every $N \times t$ sub-array includes at least once all the ordered subsets from v symbols of size t (t-tuples) (Colbourn, 2004). The minimum N for which a CA$(N; t, k, v)$ exists is the *covering array number* and it is defined according to the following expression: CAN$(t, k, v) = min\{N : \exists\, \text{CA}(N; t, k, v)\}$.

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

In practice, algorithms' input parameters do not have exactly the same number of values (levels). To overcome this limitation of CAs, mixed level covering arrays (MCAs) are used.

A MCA$(N; t, k, (v_1, v_2, \cdots, v_k))$ is an $N \times k$ array on v symbols, where each column i ($1 \leq i \leq k$) of this array contains only elements from a set S_i, with $|S_i| = v_i$. This array has the property that the rows of each $N \times t$ sub-array cover all t-tuples of values from the t columns at least once. Next, we present the details of the tuning process, based on *CIT*, for the particular

case of our SAMPARS algorithm.

First, we have identified $k = 8$ input parameters used for SAMPARS: neighborhood function $\mathcal{N}$, maximum number of visited neighboring solutions MCL, stop condition SC, reheating factor β, cooling factor α, initial temperature T_i, initial solution procedure IS and number of reheats NR. Based on some preliminary experiments, certain reasonable values were selected for each one of those input parameters (shown in Table 3.6).

Table 3.6 – Input parameters of the SAMPARS algorithm and their selected values.

NR	IS	T_i	α	β	SC	MCL	$\mathcal{N}$
0	Random	6	0.99	1.4	Final temperature	Medium	SPR
1	Greedy	$\sqrt{n}$	0.97	1.6	# max. of solutions that don't improve	Big	TBR
2	NJ	$\sqrt[2.5]{n}$	0.95	-	-	-	-
3	NJ + Fitch	$\sqrt[3]{n}$	-	-	-	-	-
4	-	-	-	-	-	-	-

The smallest possible mixed level covering array MCA$(240; 4, 8, (2, 2, 2, 2, 3, 4, 4, 5))$, shown (transposed) in Table 3.7, was obtained by using the Memetic Algorithm reported in (Rodriguez-Tello and Torres-Jimenez, 2010). As reference, we can map 0 in the last column (the first line in Table 3.7) to SPR and 1 to TBR. The resulting interaction test-suite contains, thus, 240 test cases (parameter settings) which include at least once all the 8-way combinations between SAMPARS's input parameters and their values [1]

Each one of those 240 test cases was used to execute the SAMPARS algorithm 30 times over the 20 instances of the test-suite described in Section 3.3.1.

From the results obtained we have selected the 5 test cases which yield the best results. Their average parsimony cost and the average CPU time in seconds are presented in Table 3.8. This table allowed us to observe that the parameter setting giving the best trade-off between solution quality and computational effort corresponds to the test case number 59 (shown in bold). The best average parsimony cost with an acceptable speed is thus reached with the following input parameter values: number of reheat $NR = 4$, initial solution procedure $IS = NJ$, initial temperature $T_i = 6.0$, cooling factor $\alpha = 0.99$, cooling factor $\beta = 1.6$, stop condition $SC = T_f$, maximum number of visited neighboring solutions $MCL = Big$, neighborhood function $\mathcal{N} = SPR$. These values are thus used in the experimentation reported next.

3.3.3 SAMPARS compared to an existing SA implementation

For this experiment a subset of six representative benchmark instances, taken from the test-suite described Section 3.3.1, was selected (comparable results were obtained with all the other tested instances). Then, the latest version of LVB was obtained, compiled and executed on our computational platform using the input parameters suggested by the author (Barker, 2012).

Table 3.9 displays the detailed computational results produced by this experiment. The first three columns in the table indicate the name of the instance as well as its number of taxa (n) and length (k). For each compared algorithm the best (B), average ($Avg.$), and standard deviation ($Dev.$) of the parsimony cost attained in 30 independent executions and its average CPU time in seconds are listed in columns 4 to 11. A statistical significance analysis was performed for this experiment. First, *D'Agostino-Pearson's omnibus* K^2 test was used to evaluate the normality of data distributions. For normally distributed data, either *ANOVA* or the *Welch's t* parametric tests were used depending on whether the variances across the samples were homogeneous (*homoskedasticity*) or not. This was investigated using the *Bartlett's* test. For non-normal data, the nonparametric *Kruskal-Wallis* test was adopted. A significance level of 0.05 has been considered. The resulting $P\text{-}value$ is presented in Column 12.

From Table 3.9 we can observe that SAMPARS is the most time-consuming algorithm, since it uses an average of 539.57 seconds for solving these six instances. On the contrary, LVB employs only 288.16 seconds. However, we can also remark that SAMPARS can take advantage of its longer executions. Indeed it is able to consistently improve the best results found by LVB,

1. In contrast, with an exhaustive testing which contains $5(4^2) \times 3(2^4) = 3840$ test cases.

Table 3.7 – Mixed level covering array MCA$(240; 4, 8, (2, 2, 2, 2, 3, 4, 4, 5))$ representing an interaction test-suite for tuning SAMPARS (transposed).

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
0	2	4	3	3	2	3	0	2	1	3	3	0	3	0	0	4	1	0	3	4	1	0	4	0
0	2	3	2	3	3	2	3	2	1	2	2	2	0	3	3	1	1	0	0	0	2	0	3	0
0	3	2	3	2	0	0	2	2	2	1	1	0	3	2	3	3	0	1	1	0	0	1	2	0
1	2	0	1	1	0	2	2	0	1	0	1	0	2	0	0	1	1	1	0	0	1	0	1	0
1	0	1	1	0	0	1	1	0	1	1	1	0	0	0	0	0	0	1	0	1	1	0	0	0
0	1	1	1	1	0	0	1	0	0	1	1	0	0	0	0	0	1	1	1	1	0	1	0	1
1	0	1	0	1	0	0	0	1	1	1	0	0	1	1	0	1	1	1	0	0	1	0	1	0
1	1	1	1	0	1	0	0	0	1	1	0	0	1	1	1	1	0	1	1	1	1	1	0	1

26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50
4	0	0	3	1	3	0	0	3	2	1	3	1	0	4	3	2	4	3	2	1	0	3	0	1
1	0	1	2	3	1	0	1	3	3	3	0	1	2	3	3	0	2	0	0	0	1	2	1	1
0	1	3	2	1	1	3	2	2	3	3	0	3	1	1	0	0	0	1	3	1	2	1	1	2
2	0	1	1	0	2	1	2	2	1	2	0	1	0	1	0	0	0	1	2	1	0	1	1	0
0	1	0	1	1	1	0	1	0	1	1	1	0	0	0	1	1	1	0	1	1	1	0	0	1
0	0	0	1	0	1	1	0	1	1	0	0	1	1	0	1	0	0	0	1	1	0	0	0	1
1	0	1	1	1	0	0	1	0	0	1	0	0	1	1	1	1	1	0	0	0	1	0	0	0
0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1	1	1

51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	67	68	69	70	71	72	73	74	75
2	4	3	4	0	4	1	4	4	3	1	2	3	0	0	3	1	2	0	0	2	0	0	2	1
1	1	1	3	0	1	0	3	3	1	2	3	2	1	2	2	1	3	1	1	3	2	3	0	3
3	1	3	3	2	2	0	0	0	0	1	2	0	0	3	2	1	1	1	0	3	3	1	1	1
1	2	0	2	1	1	2	2	0	1	1	1	1	2	1	2	0	2	2	0	2	0	2	1	1
0	0	1	0	0	0	1	1	1	1	0	0	0	1	1	1	0	1	0	0	0	0	1	0	0
1	1	0	0	0	1	0	0	0	0	1	0	1	0	1	0	1	1	1	1	0	0	0	0	1
1	0	0	1	1	0	0	0	1	0	0	1	1	0	1	1	1	1	1	0	1	0	0	0	1
0	1	1	0	1	1	0	0	0	0	0	1	0	0	1	1	0	1	1	1	1	1	1	0	1

76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
2	2	3	1	0	0	1	0	4	2	2	0	1	4	2	2	1	1	4	4	1	0	3	3	4
3	3	0	3	1	0	1	3	0	0	2	1	3	0	3	2	2	0	2	0	3	3	0	2	0
0	2	2	0	1	3	2	0	1	1	1	1	2	2	2	2	2	1	1	2	0	2	3	2	0
2	0	2	0	1	0	2	1	0	2	1	0	0	0	2	2	2	0	0	1	1	1	1	0	2
0	1	1	0	1	1	0	0	0	1	1	0	0	0	0	1	1	1	1	0	1	1	1	0	1
0	0	1	1	1	1	1	0	1	0	0	0	0	1	1	1	0	0	1	0	0	1	0	0	0
0	0	0	1	0	1	0	1	0	1	1	1	0	1	1	1	0	1	0	1	0	0	1	0	1
0	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	1	0	0	0	1	0	0	1	0

101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121
1	4	2	3	0	3	2	3	0	1	0	3	1	4	3	4	2	3	4	1	3
2	1	2	3	2	3	1	1	2	0	1	1	3	0	2	3	0	0	1	0	0
2	0	1	1	2	1	1	3	3	2	3	3	3	1	1	3	0	3	1	3	2
1	1	0	0	2	2	1	2	2	1	2	1	0	1	2	1	2	0	1	1	1
0	1	0	0	0	0	1	0	1	0	1	0	1	1	0	1	1	0	1	1	1
1	1	0	1	1	0	1	1	1	0	0	1	0	0	1	1	0	1	1	0	0
1	0	1	0	1	0	1	1	0	0	0	0	0	0	1	0	0	1	0	0	0
0	0	0	1	1	1	0	1	1	1	0	0	1	1	0	1	0	0	1	1	1

122	123	124	125	126	127	128	129	130	131	132	133	134	135	136	137	138	139	140	141	142
0	0	4	4	3	1	3	0	4	4	0	1	2	0	3	1	2	1	4	2	3
3	0	2	2	0	0	2	2	0	1	1	0	3	2	1	1	3	1	3	1	3
3	2	1	2	1	2	3	1	3	3	2	0	1	0	2	1	3	1	2	1	2
2	2	1	0	0	2	0	2	0	2	0	0	0	1	1	2	0	1	2	2	0
0	0	0	0	1	1	0	1	1	0	1	1	0	0	0	1	0	0	0	0	1
1	1	0	1	0	1	0	0	1	0	1	1	1	1	1	0	0	1	0	0	0
1	0	0	0	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	0	0	0	1	0	1	0	0	0	1	0	1	0	1	0	0	1	0	1

143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160	161	162	163
1	4	1	4	1	2	4	4	2	1	2	3	2	2	2	3	0	3	0	4	3
3	3	1	2	1	2	2	3	0	2	0	1	3	0	2	3	3	1	1	2	0
2	3	0	2	3	0	1	1	0	3	1	0	0	3	0	0	3	2	0	0	2
2	0	0	1	0	1	2	0	1	2	0	2	1	1	0	1	1	2	1	2	0
1	0	1	1	0	1	0	1	0	0	1	1	1	1	0	0	1	1	1	0	0
1	1	0	0	1	1	0	0	1	0	1	0	1	1	0	0	0	0	0	0	0
1	1	0	1	1	0	1	0	0	0	0	0	1	0	0	0	1	1	0	1	0
1	1	0	0	1	1	1	0	1	0	0	1	0	1	1	0	0	0	1	1	0

164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184
1	2	2	1	0	1	2	1	0	1	0	1	2	4	3	0	1	2	3	1	1
2	2	2	1	1	3	1	0	3	2	1	0	0	3	1	3	3	2	3	3	0
0	1	0	3	2	2	0	1	1	2	3	3	2	0	2	0	3	2	1	1	3
2	2	2	2	1	1	0	2	1	0	0	0	0	1	0	0	1	1	1	2	2
0	1	0	1	0	1	0	0	0	0	1	0	1	1	1	1	0	0	1	0	1
0	1	0	0	1	1	1	1	0	0	1	1	1	1	1	1	1	0	0	1	1
1	0	1	1	1	1	0	0	1	0	1	1	0	1	0	1	1	0	1	0	0
1	0	0	1	1	0	0	0	1	1	0	0	0	1	0	1	1	1	1	0	0

185	186	187	188	189	190	191	192	193	194	195	196	197	198	199	200	201	202	203	204	205
4	4	2	3	3	4	4	1	4	0	1	1	0	0	2	4	1	2	3	0	4
2	0	1	0	2	2	0	2	0	3	0	2	0	3	1	2	1	1	1	2	1
0	1	1	0	0	3	3	3	2	1	0	0	2	0	3	3	0	3	1	2	2
1	2	0	2	0	0	2	1	2	0	1	0	2	2	2	1	2	0	0	1	0
1	0	1	0	1	1	1	1	1	1	0	0	1	1	1	0	0	1	1	0	0
0	1	0	1	1	1	0	1	1	1	0	1	0	0	0	1	0	1	1	0	0
0	1	0	0	1	1	1	1	0	0	1	0	1	0	0	0	1	1	1	1	1
0	1	0	1	1	0	1	1	0	0	1	0	1	1	1	1	1	1	1	0	0

206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226
0	2	4	2	0	4	2	3	1	4	2	4	4	2	2	4	2	3	3	1	1
2	0	1	0	2	2	1	3	2	2	2	3	0	3	2	1	1	0	1	3	2
0	2	0	3	2	3	2	3	1	2	3	0	3	1	3	3	2	0	0	0	1
2	2	0	0	0	2	0	2	2	2	0	0	1	1	1	0	1	1	0	1	0
1	0	0	0	1	1	0	1	1	0	1	0	0	0	1	1	1	0	0	0	1
0	0	0	0	0	1	0	1	1	1	1	1	1	0	0	1	0	1	1	1	0
0	0	0	1	0	1	1	0	1	1	0	0	0	1	1	0	1	1	1	1	0
0	1	1	1	1	0	1	1	0	1	0	0	0	1	0	1	0	0	1	1	1

227	228	229	230	231	232	233	234	235	236	237	238	239	240
2	2	0	4	3	3	4	3	4	2	2	3	4	0
0	1	0	1	2	3	0	3	3	1	1	3	1	0
2	2	0	1	3	3	0	3	1	0	0	0	2	0
1	2	2	0	2	1	1	0	2	2	1	2	2	0
0	1	0	0	1	1	0	0	1	1	1	1	1	1
1	1	0	0	0	0	0	0	1	1	0	1	0	1
1	0	0	1	1	0	1	1	0	1	0	1	0	1
0	1	0	0	0	0	0	1	1	1	0	1	1	0

obtaining in certain instances, like *tst08*, an important decrease in the parsimony cost (up to $-15 = 852 - 867$). Furthermore, the solutions found by SAMPARS present a relatively small standard deviation (see column $Dev.$). It is an indicator of the algorithm's precision and robustness since it shows that in average the performance of SAMPARS does not present important fluctuations. We have noticed that LVB has a erratic behavior (see Table 3.10).

Table 3.8 – Results from the 5 best parameter test cases in the tuning experiment.

Num.	Test case	Avg. parsimony	Avg. CPU time
59	**43001010**	**1004.06**	**3512.51**
233	40010010	1004.14	3511.35
217	43000100	1005.00	2058.93
37	30001000	1005.02	2047.52
118	30300101	1005.29	3136.40

Table 3.9 – Comparison between SAMPARS and LVB for a subset of six selected instances. All tests were significant at the 0.05 level

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

For example, the resolution of problem *tst04* can take from 84 seconds to 5 hours and the time spent for the resolution is not correlated to the finding of a better solution.

Table 3.10 – Erratic results of LVB on instance *tst04* for 5 executions

time (s)	score	number of trees
84	609	16
13194 (>3h)	595	4747
19532 (>5h)	600	30
1960 (32m)	605	92
307 (5m)	609	510

The statistical analysis presented in the last two columns of Table 3.9 confirms that there exist a statistically significant increase in performance achieved by SAMPARS with regard to LVB on the six studied instances. Thus, we can conclude that SAMPARS is more effective than the existing SA algorithm reported in (Barker, 2003, 2012).

Another implementation called *Tree-Drifting* is described in (Goloboff, 2002) as a TBR descent able to accept suboptimal trees. The key component of the method is the function designed to determine the probability of acceptance, which is based on both the absolute step difference and a measure of character conflict: the *relative fit difference*, which is the ratio of steps gained and saved in all characters, between the two trees. A pseudocode of the algorithm is given in (De Laet, 2005). Tests performed using only tree-drifting in TNT (Goloboff, Farris, and Nixon, 2008a) gave results far from the best known values (results not presented here).

3.3.4 Comparison of SAMPARS with the State-of-the-art procedures

In this experiment a performance comparison of the best solutions achieved by SAMPARS with respect to those produced by GA+PR+LS (Ribeiro and Vianna, 2009), TNT (Goloboff, Farris, and Nixon, 2008a) and Hydra (Goëffon, 2006) was carried out over the test-suite described in section 3.3.1.

TNT (*Tree analysis using New Technology*) is probably the fastest and one of the most effective parsimony analysis program for the MP problem. TNT is known for finding better trees several thousands times faster than other software. TNT uses many search strategies (Goloboff, 2002) coming from genetic algorithms, local search and supertrees. During the local search phases

based on SPR and TBR, TNT can visit millions of trees in a very short time as it is based on Gladstein's *incremental down-pass optimization* (Gladstein, 1997).

The results from this experiment are depicted in Table 3.11. Columns 1 to 3 indicate the instance and its size in terms of taxa (n) and length (k). The best solutions found by GA+PR+LS , TNT and Hydra, in terms of parsimony cost Φ are listed in the next three columns. Note that for TNT this corresponds to the best result over 30 runs. Columns 7 to 9 present the best (B), average ($Avg.$) and standard deviation ($Dev.$) of the parsimony cost attained by SAMPARS in 30 independent executions, as well as its average CPU time in seconds (Column 10). Finally, the difference (δ) between the best result produced by the SAMPARS algorithm and the best-known solution produced by either GA+PR+LS or Hydra is shown in the last column.

Table 3.11 – Performance comparison among SAMPARS, GA+PR+LS, TNT and Hydra over 20 standard benchmark instances.

						SAMPARS				
Instance	n	k	GA+PR+LS	TNT	Hydra	B	$Avg.$	$Dev.$	$Time$	δ
tst01	45	61	547	545	545	545	545.13	0.43	1407.57	0
tst02	47	151	1361	1354	1354	1354	1355.30	0.97	1938.23	0
tst03	49	111	837	834	833	833	833.43	0.56	2506.30	0
tst04	50	97	590	589	588	**587**	588.23	0.80	1341.17	-1
tst05	52	75	792	789	789	789	789.00	0.00	2007.90	0
tst06	54	65	603	597	596	596	596.57	0.56	1164.27	0
tst07	56	143	1274	1273	1269	1269	1270.83	1.63	4063.80	0
tst08	57	119	862	856	852	852	853.33	1.27	2884.73	0
tst09	59	93	1150	1145	1144	**1141**	1144.73	1.09	3237.53	-3
tst10	60	71	722	721	721	**720**	720.80	0.70	2288.00	-1
tst11	62	63	547	543	542	**541**	542.21	0.72	3807.79	-1
tst12	64	147	1225	1219	1211	**1208**	1215.27	2.76	3668.40	-3
tst13	65	113	1524	1516	1515	1515	1517.77	1.91	2514.20	0
tst14	67	99	1171	1162	1160	1160	1163.03	1.82	2847.13	0
tst15	69	77	758	755	752	752	753.90	1.11	4808.63	0
tst16	70	69	537	531	529	529	531.00	1.23	3268.20	0
tst17	71	159	2469	2453	2453	**2450**	2456.00	2.63	8020.23	-3
tst18	73	117	1531	1522	1522	**1521**	1525.67	3.96	4451.37	-1
tst19	74	95	1024	1017	1013	**1012**	1016.23	2.14	6875.30	-1
tst20	75	79	671	666	661	**659**	662.82	1.44	7149.43	-2
Avg.			1009.75		1002.45	1001.65	1004.06	1.39	3512.51	

The analysis of the data presented in Table 3.11 lead us to the following observations. First, we clearly see that the procedure GA+PR+LS (Ribeiro and Vianna, 2009) consistently returns poorer quality solutions than Hydra and SAMPARS. Second, the best solutions attained by the proposed SAMPARS algorithm are very competitive with respect to that produced by the existing state-of-the-art procedure Hydra (Goëffon, 2006), since on average SAMPARS provides solutions whose parsimony costs are smaller (compare columns 6 and 7). In fact, it is able to improve 9 previous best-known solutions produced by Hydra and to reach the results of the other 11 benchmark instances.

Thus, as this experiment confirms, our SAMPARS algorithm is an effective alternative for solving the MP problem, compared with the three representative state-of-art algorithms: GA+PR+LS, TNT and Hydra in terms of the quality of the solution.

3.4 Conclusions

In this chapter we have presented an improved Simulated Annealing algorithm called SAMPARS to find solutions for the MP problem under the optimality criterion of Fitch. SAMPARS's components and parameter values were carefully determined,

through the use of a tuning methodology based on Combinatorial Interaction Testing (Cohen, Dalal, Parelius, and Patton, 1996), to yield the best solution quality in a reasonable computational time.

An extensive experimentation was conducted to investigate the performance of SAMPARS over a set of 20 well-known benchmark instances. In these experiments our algorithm was carefully compared with an existing Simulated Annealing implementation (LVB) (Barker, 2003, 2012), and other three state-of-the-art algorithms GA+PR+LS, TNT and Hydra. The results show that there exists a statistically significant increase in performance achieved by SAMPARS with respect to LVB. SAMPARS is in fact able to consistently improve the best results produced by LVB, obtaining in certain instances important reductions in the parsimony cost. Regarding GA+PR+LS (Ribeiro and Vianna, 2009), we have observed that on average this algorithm returns worse quality solutions than SAMPARS. Compared with the state-of-the-art algorithm called Hydra (Goëffon, 2006) our SAMPARS algorithm was able to improve 9 previous best-known solutions and to equal the results on the other 11 selected benchmark instances. Furthermore, it was observed that the solution cost found by SAMPARS presents a relatively small standard deviation, which indicates the precision and robustness of the proposed approach. These experimental results confirm the practical advantages of using our algorithm for solving the MP problem.

4

A Bottom-up implementation of Path-relinking

4.1 Introduction

Path Relinking (PR), has proved unusually effective for solving a wide variety of optimization problems from both classical and real world settings. PR operates with a population of solutions, rather than with a single solution at a time, and employs procedures for combining these solutions to create new ones (Glover et al., 2000).

In this chapter we describe a bottom-up implementation of Path-Relinking for phylogenetic trees. This bottom-up implementation is compared to two versions of an existing top-down implementation. We show that our implementation is more efficient, more interesting to compare trees and to give an estimation of the distance between two trees in terms of the number of transformations.

4.2 Path-relinking

Path-relinking (PR) is a metaheuristic with an intensification strategy used to explore elite solutions. It was defined by Glover et al. (2000) and it is closely related to Tabu Search. The main objective of PR is to generate paths between and beyond a group of solutions by combining them, where solutions on such paths also serve as sources for generating additional paths. In order to get good paths, the solutions to combine must be selected in a neighborhood space, rather than in Euclidean space (Glover, 1998).

PR has a group of guiding solutions with weighted attributes to determine which moves are given higher priority. The generation of such paths in neighborhood space characteristically *relinks* previous points in ways not achieved in the previous search history, hence giving the approach its name.

Given two solutions called source and guiding, PR consists in transforming the source solution into the guiding solution by applying a series of modifications. After each modification an exploration phase is performed on a copy of the current solution. The aim of PR is to generate a path from the source to the guiding solution in order to possibly find a better solution.

PR has been used in different domains for the resolution of combinatorial optimization problems (Wang et al., 2012; Seridi et al., 2013; Zhou et al., 2013; Resende and Ribeiro, 2014).

Ribeiro and Vianna (2009) describe an implementation of PR tailored to trees for the resolution of the Maximum Parsimony problem (see algorithm 11). The symmetric difference $\Delta(x, y)$ computes the set of moves to go from solution x to solution y. At each step the algorithm selects $m^\star$ the move that minimizes the fitness function from the current solution s_i to t. If the new solution $s_{i+1} = s_i \oplus m^\star$ is better than $s^\star$ then $s^\star$ is updated. On randomly generated instances they observed that using GRASP + PRm they could always find better solutions than a simple GRASP algorithm.

Algorithm 11: Path Relinking from Ribeiro and Vianna (2009)

PathRelinking(s, t)
input: s, t : trees such that $fitness(s) >= fitness(t)$
output: best solution found $s^\star$
$s^\star \leftarrow s$ (by definition)
$i \leftarrow 0$
$s_i \leftarrow s$
while $\Delta(s_i, t) \neq 0$ **do**
  $m^\star \leftarrow argmin\{fitness(x \oplus m) : m \in \Delta(s_i, t)\}$
  $s_{i+1} \leftarrow s_i \oplus m^\star$
  if $fitness(s_{i+1}) < fitness(s^\star)$ **then**
    $s^\star \leftarrow s_{i+1}$
  $i \leftarrow i + 1$
return $s^\star$

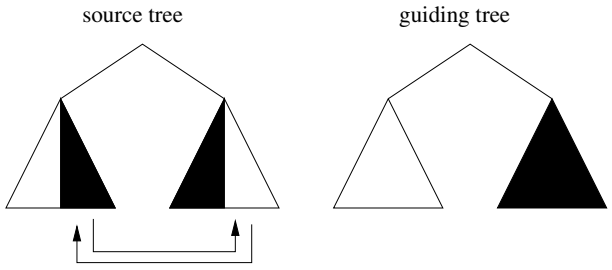
The implementation of Ribeiro and Vianna (2009) can be called *top-down*: it starts from the root of the tree and recursively explores each left and right subtree. For each iteration they compare the taxa in the left and right subtrees of the source and guiding solutions. All taxa of the source left (resp. right) subtree that are in the right (resp. left) subtree of the guiding solution are moved to the right (resp. left) subtree of the source solution. The major drawback of this implementation is that it requires a lot of modifications and moves of the taxa from left to right (resp. right to left) subtrees. When a taxon is degraphed it is then regraphed on a branch of the sibling subtree that minimizes the parsimony score of the tree. We can also put the degraphed taxon at the top of the sibling subtree to avoid the search of a minimum tree. If we remove the exploration phase from the PR algorithm we then have an algorithm that transforms one tree into another which can be used to compare trees and define a measure of distance between them. This measure of distance seems interesting because it is closer to the topology of the trees to compare.

In the next section we describe the method that we have designed to perform Path-Relinking between two trees based on the difference of subtrees. Then we show the results and analysis of our experimentation.

4.3 The Bottom-up implementation

Ribeiro and Vianna (2009) implemented PR to solve the MP problem by using a recursive algorithm that starts from the root of the tree and compares the left and right subtrees of the source and guiding solutions. All taxa of the left subtree of the source that are present in the right subtree of the guiding solution are moved to the right subtree of the source solution and conversely (see Figure 4.1).

Figure 4.1 – Top-Down implementation



The major drawback of this implementation is that it requires a lot of modifications and moves of the taxa from left (resp. right) to right (resp. left) subtrees.

In contrast, we have implemented a **bottom-up iterative** solution which compares the subtrees present in the source and guiding solutions (see Figure 4.2). For this the subtrees of each solution are ordered by their number of leaves and we start to compare subtrees of size 2, then subtrees of size 3, and so on (see Algorithm 12).

When a subtree of the guiding solution $t = (X, Y)$ is not found in the source solution, we have to transform the subtree in the source solution into the one in the guiding solution. Here X is the left subtree of t and Y its right subtree.

Algorithm 12: Path-Relinking with bottom-up iterative implementation

input: s: source tree, g : guiding tree
output: number of transformations

```
1  reorder(s) ;
2  reorder(g) ;
3  transformations ← 0;
4  Ω_g ← ordered set of subtrees of guiding tree g;
5  change ← true;
6  while change do
7      Ω_s ← ordered set of subtrees of source tree s;
8      change ← false ;
9      if ∃  t = (X, Y) ∈ Ω_g \ Ω_s then
10         change ← true ;
11         degraph Y and regraph on X in s;
12         transformations ← transformations + 1;
13 return transformations
```

Table 4.1 – Sets of subtrees in the source and guiding trees ordered by number of leaves

#	Source	Guiding
1	A, B, C, D, E, F	A, B, C, D, E, F
2	(A, F), (C, E), (B, D)	(A, B), (E, F)
3		$((A, B), C)$, $(D, (E, F))$
4	$((A, F), (C, E))$	
5		
6	$(((A, F), (C, E)), (B, D))$	$(((A, B), C), (D, (E, F)))$

Consider the example of Figure 4.2 (a) where we can see the source and guiding trees. A preliminary modification of the trees (line 1 and 2 of the algorithm) consists in reordering the trees by the lexicographic order of the leaves in order to be able to efficiently compare the subtrees. In fact the subtrees (A, B) and (B, A) are equal and we do not want to have to check both cases, so we will only allow subtrees of the form (A, B).

On each node, one of the sequences on the left subtree must be inferior to all the sequences on the right subtree. For example, the node (B, A) will be changed by swapping the leaves in order to obtain (A, B). The subtree $(C, (A, B)))$ will be reordered as $((A, B), C)$ because on the right node, A and B are inferior to C. The subtree $((C, E), (A, F))$ will be reordered as $((A, F), (C, E))$ because A that initially appears on the right subtree is inferior to C and E.

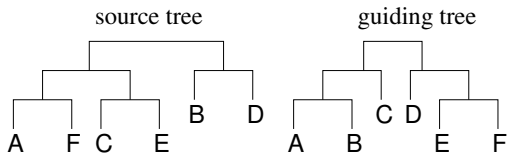
The subtrees present in the source and guiding solutions are represented in Table 4.1 in Newick notation. After reordering the source and guiding trees we can enter the main loop of the algorithm (lines 6 to 12).

The subtree (A, B) present in the guiding tree is not present in the source tree so we degraph B from the source tree, and regraph it on A (see Figure 4.2(b)). On Figure 4.2(b) the subtree (E, F) present in the guiding tree is not present in the source tree so we degraph F from the source tree and regraph it on E.

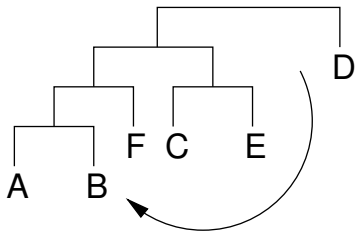
On Figure 4.2(c) the subtree $((A, B), C)$ present in the guiding tree is not present in the source tree, we have to degraph C from the source tree and regraph it on (A, B).

The subtree $(D, (E, F))$ present in the guiding tree is not present in the source tree: degraph (E, F) from the source tree and regraph it on D (see Figure 4.2(e)).

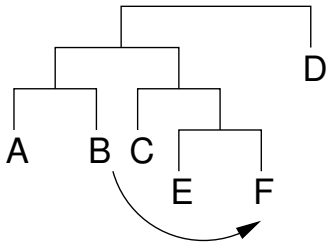
Finally the source and guiding trees are equal so we can stop the algorithm.



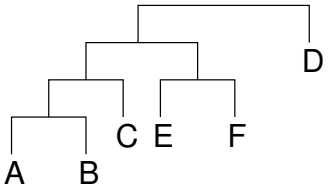
(a) source and guiding trees



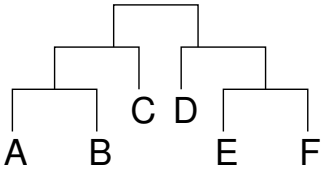
(b) first modification resulting in (A, B) on source



(c) modification resulting in (E, F)



(d) modification resulting in $((A, B), C)$



(e) modification resulting in $(D, (E, F))$

Figure 4.2 – Example of Bottom-Up Path-Relinking with source and guiding trees

4.4 Complexity of PR-Bottom-up

The complexity of the algorithm can be computed as follows: given n, the number of taxa of the problem, the reordering of the guiding and source trees needs $n - 1$ comparisons and the computation of Ω_g can be done in $2n - 1$ operations. The main loop will be executed a certain number of times, let's say p times, and we will need to compute Ω_s, find a missing subtree (the maximum will be n comparisons) and perform a degraph and regraph (1 transformation). This adds up to:

$$2 \times (n-1) + 2n - 1 + p \times (2n - 1 + n + 1) = 4n - 3 + p \times (3n) \simeq 3n \times p \tag{4.1}$$

In the worst case $p = n$, so the worst case complexity is $O(n^2)$ for the *bottom-up* implementation.

For the *top-down* implementation, the computation of the complexity is more difficult to establish as the process is recursive and the size of the subtrees changes. At each step we must compute the sets of leaves on the left and right subtrees of the source and guiding solutions. Then move the leaves in the source tree from left to right or from right to left. The number of transformations (see the tables in the results section) can give us some insight into the complexity of this implementation.

4.5 Experimentation and results

4.5.1 Benchmark

We used the instance called zilla (500 sequences of 759 DNA residues) that was originally obtained from the chloroplast gene rbcL (Chase, Soltis, Olmstead, Morgan, Les, Mishler, Duvall, Price, Hills, Qiu, Kron, Rettig, Conti, Palmer, Manhart,

Sytsma, and al, 1993) and the best parsimony score of 16,218 was first found by TNT. In Table 4.2 we report the percentage of common subtrees that we could find between two trees that are being compared. The number of common subtrees can be thought as a measure of similarity between trees. Those different trees were obtained using SAMPARS.

For example two trees of score 16,218 (namely 16,218 and $16{,}218^b$) have 74.75% subtrees in common which means that they are topologically close. On the contrary the tree of score 21,727 has 0% subtrees in common with the tree of score 16,218. This means they are very far from each other and have only the leaves in common.

Table 4.2 – Percentage of common subtrees for Parsimony trees of zilla used for Path-Relinking

source / guiding	% common subtrees
16218^b / 16218	74.75%
16219 / 16218	73.15%
16250 / 16218	61.12%
16401 / 16218	45.69%
16611 / 16218	36.27%
21727 / 16218	0.00%
16250 / 16219	61.72%
16401 / 16219	48.70%
16401 / 16250	45.49%
16611 / 16250	44.89%
21727 / 16250	0.20%
16611 / 16401	36.67%
21727 / 16611	0.20%

Three different implementations were compared:

— *bottom-up*: iterative implementation explained in this chapter, based on subtree difference,
— *top-down without minimization*: recursive implementation without optimization on regraph, i.e. when a leaf is regraphed it is placed at the root of the subtree where it should appear (Ribeiro and Vianna, 2009),
— *top-down with minimization*: recursive implementation with optimization on regraph, i.e. when a leaf is regraphed all possible branches are tested and we keep the first one that minimizes the score of the tree (Ribeiro and Vianna, 2009).

4.5.2 Experiments

The experimentations were performed on an Intel Core i5 4570 and the program was coded in Java 1.7, it is part of a software called Arbalet [1]. In Table 4.3 we report for each implementation the number of transformations (degraph + regraph), the execution time in seconds, the number of times the source tree had a score inferior or equal to the guiding tree (#Equal) and the number of times the source tree had a score strictly inferior to the guiding tree (#Less) during the generation of the path. Note that the source tree has a higher score than the guiding tree. This is not necessary for the *bottom-up* method for which we can invert the trees. However this is required by the *top-down with minimization* implementation.

1. http://www.info.univ-angers.fr/pub/richer/ur.php?arbalet

Table 4.3 – Results of Path-Relinking for the different implementations (Times in seconds)

	Bottom-Up				Top-Down No Minimization				Top-Down With Minimization			
source / guiding	**Trans.**	**Time**	**#Equal**	**#Less**	**Trans.**	**Time**	**#Equal**	**#Less**	**Trans.**	**Time**	**#Equal**	**#Less**
16218^b / 16218	24	0.10	13	0	118	0.33	16	0	74	0.85	21	0
16219 / 16218	32	0.14	2	0	462	1.36	1	0	343	4.08	3	0
16250 / 16218	97	0.40	3	0	1770	4.79	1	0	1519	25.32	2	0
16401 / 16218	151	0.60	2	0	1891	5.16	1	0	1474	25.36	2	0
16611 / 16218	186	0.75	2	0	1903	5.19	1	0	1225	19.57	2	0
21727 / 16218	446	1.68	2	0	1881	5.17	1	0	1241	18.70	2	0
16250 / 16219	92	0.37	2	0	1722	4.76	1	0	1418	24.70	1	0
16401 / 16219	152	0.61	2	0	1867	6.17	1	0	1390	23.74	1	0
16401 / 16250	162	0.63	1	0	1707	4.71	1	0	1203	19.74	1	0
16611 / 16250	144	0.56	1	0	1746	4.79	1	0	1233	35.78	3	0
21727 / 16250	449	1.71	1	0	1712	4.77	1	0	1196	19.65	1	0
16611 / 16401	202	0.79	3	0	1602	4.42	1	0	1216	47.75	2	1
21727 / 16611	455	1.84	2	0	1842	5.16	1	0	1528	22.40	593	586

In Table 4.3, with the *bottom-up* implementation the path between the two trees of score 16,218 (called 16,218 and $16,218^b$) was built with 24 transformations in 0.1 seconds. As mentioned before those trees are topologically very close. During the transformation process the source tree, when modified, had a best score (of 16,218) 13 times among the 24 transformations. With the *top-down with minimization* algorithm (see results of Table 4.3) the construction of the path of the tree of score 21,727 into the tree of score 16,611 has needed 1528 transformations and took 22.4 seconds. It also lead to the generation of 586 trees of score under 16,611. This means that the *top-down with minimization* algorithm can sometimes help find a tree of lower score than the guiding tree.

4.5.3 Execution time

Although the *bottom-up* implementation requires to recompute the set of subtrees of the source tree for each iteration it is the fastest method. The *top-down with minimization* implementation takes much more time because of the optimization phase on regraph. However for the *top-down* implementations the number of transformations in terms of nodes to degraph and regraph is very important compared to the *bottom-up* version. For example for the search of a path from tree 21,727 to 16,218, the number of recursive calls is equal to 499. The number of leaves moved for the first 10 steps of the recursion are: 194, 143, 21, 3, 15, 34, 40, 4, 43, 3, ... for a total of 1881 transformations.

4.5.4 Comparison of trees

In the *bottom-up* implementation the number of transformations is proportional to the topological modifications of the tree. For example, on Table 4.3 the number of transformations from the tree of score 16,219 to 16,218 is 32, while the number of transformations from the tree of score 21,727 to 16,218 is 455. For the *top-down implementation without minimization* the transformation from the tree of score 16,401 to 16,218 is nearly equal to the number of transformations of the tree of score 21,727 to 16,218. Based on this results we could say that our implementation is more suitable to compare trees than the *top-down* of Ribeiro and Vianna (2009).

There exists different metrics and methods to compare trees. One of the most famous is the Robinson-Foulds metric (RF) (Robinson and Foulds, 1981) initially designed for unrooted tree but which has a variant for rooted trees based on clusters (Lin, Rajan, and Moret, 2012). RF is equal to the number of different splits in compared trees. A split $A|B$ of a set L is an unordered

pair (ie, $A|B = B|A$) of its subsets, such that $L = A \cup B$ and $A \cap B = \emptyset$. Our method can to some extent be brought closer to the Matching Cluster metric for rooted trees (Lin, Rajan, and Moret, 2012) but provides different results.

4.5.5 Ability to find better solutions

When PR does not use the exploration phase it is unable to find a tree with a score lower than the guiding tree (see columns #Less and #Equal), except for the *top-down with minimization* version but only for trees that are far from the best known solution.

Nevertheless it is possible to add a minimization phase to the *bottom-up* algorithm: the new subtree $t = (X, Y)$ obtained from line 11 of Algorithm 12 can be degraphed and regraphed somewhere on the tree in order to minimize the parsimony score. However t must not be regraphed on some previously modified subtree in order to avoid any cycle of transformation that would cause a modification to be undone and that would be performed again at the next iteration. We have tested this method but it does not lead to any improvement and we could not find a score inferior to the score of the guiding tree.

4.6 Conclusion

Path-Relinking in its *bottom-up* implementation represents an interesting tool to compare the topologies of the source and guiding trees. The *bottom-up* iterative implementation what we have described is faster than the *top-down* recursive implementation and can serve as a measure of distance between trees and could be applied to any other context when one has to compare trees: for example when dealing with a genetic algorithm, the replacement of some individual in the population by a new individual could only be possible if the new individual is not too *close* to the individuals that constitute the population.

5

Prediction of the parsimony score

5.1 Introduction

Given a solver it is always difficult to know the combination of parameters that will provide a result close to the optimum solution that we want to find, or in other words, how could we know how far or close is the final solution we obtain from the global optimum ? In this chapter we present a predictor designed to estimate the optimum score of an instance based on a database of solved instances. As we shall see later on, knowing that the estimate of the best score is far from the random or greedy score, we have the information that the problem is probably difficult to solve and that it would be preferable to solve it using powerful techniques. On the contrary if we know that the problem is easy to solve, it is not necessary to use resolution methods and techniques that will cost a lot of computation and consequently a lot of useless time and efforts. This work is still under way.

5.2 The main idea

In the first instance, the idea of designing a predictor is not new. It comes from the fact that it seems easier to estimate a solution than to solve the entire problem. However the design of a predictor for MP is a new concept, to which little (or no) attention has yet been paid and came from an idea we had in mind and that was partially covered by a small study we made using the R mathematical tool.

We took a set of twenty problems that we call *set1* (see Table 5.1 for the description of the sets of problems) for which we had computed different initialization trees and their scores. We had generated a table with the following numerical data for each problem:

— K the number of sequences [1]
— L the length of a sequence or number of residues (all sequences have the same length)
— R the average score (over 100 runs) of initial solutions generated randomly
— N the score of the tree obtained from the neighbor-joining algorithm (only one tree)

1. Note that throughout the thesis we used N or n for the number of sequences, but in this case N will be used to identify the neighbor-joining score.

— G the average score (over 100 runs) of initial solutions generated with a greedy algorithm
— B the best known score, considered as the optimum score obtained by using TNT or SAMPARS

We have decided to use a multiple linear regression (MLR) model (Xin and Xiao, 2009), mainly for its simplicity, in order to establish a relationship between (K, L, R, N, G) and B. The principle of MLR is to have as input a matrix M and a vector Y of observed data for which we want to determine if M and Y are correlated. We want to find a vector X and constants α and ϵ such that :

$$\alpha + M.X + \epsilon = Y' \simeq Y \tag{5.1}$$

Generally linear regression models are often fitted using the least squares approach where it is necessary to minimize $|Y - Y'|^2 = \sum_i (y_i - y'_i)^2$ in order to have Y' as close as possible of Y. To be able to compute the MLR, the matrix $M(u \times p)$ will contain u instances for which we have p properties and $u \geq p$.

From the data we had gathered and computed (see Table 5.1) we could obtain a *model* with a R-square (R^2) of 0.9999 and a p-value less than 2.2e-16, which means the data are strongly correlated.

Following is the output of the program R, where `V2` is K, ..., `V6` is G and `V7` is B:

```
> data = read.table("data.rdata")
>  model <- lm(V7 ~ V2 + V3 + V4 + V5 + V6, data)
> summary(model)

Call:
lm(formula = V7 ~ V2 + V3 + V4 + V5 + V6, data = data)

Residuals:
    Min      1Q  Median      3Q     Max
-9.1111 -1.7844 -0.6903  2.1100 10.1303

Coefficients:
    Estimate Std. Error t value Pr(>|t|)
(Intercept) 65.65223   10.49157   6.258 2.10e-05 ***
V2          -1.13401    0.31890  -3.556 0.003163 **
V3          -0.52265    0.11382  -4.592 0.000419 ***
V4          -0.16641    0.11465  -1.451 0.168685
V5           0.09717    0.13636   0.713 0.487797
V6           1.08315    0.19320   5.606 6.47e-05 ***
---
Signif. codes:  0 *** 0.001 ** 0.01 * 0.05 . 0.1 ' ' 1

Residual standard error: 4.497 on 14 degrees of freedom
Multiple R-squared:  0.9999,Adjusted R-squared:  0.9999
F-statistic: 4.205e+04 on 5 and 14 DF,  p-value: < 2.2e-16
```

The first column (*Estimate*) gives us the coefficients of the formula. The last column (*Pr(>|t|)*) is the p-value and helps test the null hypothesis, i.e. check whether the corresponding variable set to zero would affect or not the result. A variable that has a low p-value is likely to be a meaningful addition to the model. A large p-value suggests that changes in the variable are not associated with changes in the response.

Although the analysis of the p-values, R^2 and F-Test are very important to find out if the data are correlated, we are only interested by the forecasting, i.e. prediction from the MLR and we will not delve into statistical details.

The average error of the prediction was of 2.78 points with a minimum of 0.14 for instance *tst05* and a maximum of 10.13 for instance *tst02* i.e. 0,74 % of its score. For the sake of simplicity, in the remaining of the chapter, we will report real numbers with two decimal places.

The formula obtained is $Y' = E(K, L, R, N, G) = 65.65 - 1.13 \times K - 0.52 \times L - 0.17 \times R + 0.09 \times N + 1.08 \times G$, where E is the estimation of the best score $Y = B$. The constant (65.65), K, L and G are reported to have a high significance.

problem	K	L	R	N	G	B	E	$B-E$
tst01	45	61	724	588	580	545	547.62	-2.62
tst02	47	151	1629	1454	1422	1354	1343.87	10.13
tst03	49	111	1064	908	894	833	831.57	1.43
tst04	50	97	789	674	645	587	591.08	-4.08
tst05	52	75	1012	845	838	789	788.86	0.14
tst06	54	65	814	651	642	596	593.62	2.38
tst07	56	143	1586	1401	1365	1269	1278.11	-9.11
tst08	57	119	1108	951	929	852	853.08	-1.08
tst09	59	93	1450	1216	1215	1144	1143.02	0.98
tst10	60	71	994	799	785	720	722.99	-2.99
tst11	62	63	772	611	598	540	541.04	-1.04
tst12	64	147	1547	1357	1309	1208	1208.50	-0.50
tst13	65	113	1917	1629	1606	1515	1511.69	3.31
tst14	67	99	1519	1257	1247	1160	1157.98	2.02
tst15	69	77	1066	832	833	752	752.87	-0.87
tst16	70	69	778	614	594	527	523.79	3.21
tst17	71	159	2975	2597	2578	2450	2451.67	-1.67
tst18	73	117	1956	1656	1630	1521	1522.66	-1.66
tst19	74	95	1380	1126	1110	1012	1014.14	-2.14
tst20	75	79	956	774	738	659	654.79	4.21

Table 5.1 – Multiple linear regression model for *set2*, $E(K, L, R, N, G) = 65.65 - 1.13 \times K - 0.52 \times L - 0.17 \times R + 0.09 \times N + 1.08 \times G$

The fact that we could obtain a formula which gives a good precision probably comes from the relative similarity of those problems: they were all generated using the same software, are composed of a small number of sequences and a small number of residues (see K and L columns).

We then tried to add more problems to the initial set of twenty problems and recomputed the MLR, but it seemed obvious that it would not work for two reasons: we needed more parameters and we needed to take into consideration problems that were *close* to each other.

For example if we take into account the problems of *set1* and *set2* for a total of 28 problems, then the average error rises to 7.61 with a minimum of 0.64 for *tst20* and a maximum of 22.01 for instance *tst15*, i.e. 2.92 % of its optimum score.

5.3 Database of problems

We have created a database of more than 12,000 problems for which we have calculated all properties cited above. The main goal is to supply enough problems close to each other in order to be able to perform the MLR. The determination of the optimum score B was made using TNT [2] (Goloboff, Farris, and Nixon, 2008a) and the results obtained with SAMPARS. We consider that the best score obtained by the software represent the global optimum. This might not be true for all instances but we take it for granted.

The database or knowledge base is composed of the following sets:

— $set1$: eight real-life instances by Luckow and Pimentel (1985)
— $set2$: twenty randomly generated instances presented in (Ribeiro and Vianna, 2005)

2. the parameters used to solve the instances are `rseed 0, bground, mxram 5096, nstates num, nstates nogaps, rep+1, mult:tbr, mult=ho3, mult=ratchet, sectsch:rss`

— $set3$: problems from Treebase [3]
— $set4$: problems from (Morrison, 2007)
— $set5$: problems taken from (White and Holland, 2011)
— $set6$: problems from (Goloboff, Carpenter, Arias, and Esquivel, 2008b) especially zilla
— $set7$: 12126 problems from TreeFam [4]
— $set8$: 2 subproblems of ribosomal RNA sequences 23S [5] from the *Empirical Datasets with Reference Topologies* [6]
— $set9$: 40 subproblems of 16S [7] ribosomal RNA trees of FastTree [8]

Table 5.2 reports the different global information about each set of problems:

— the number of instances (#instances)
— the minimum number of taxa (K_{min})
— the maximum number of taxa (K_{max})
— the minimum length for a taxon (L_{min})
— the maximum length for a taxon (L_{max})
— the minimum similarity (S_{min})
— the maximum similarity (S_{max})

set	#instances	K_{min}	K_{max}	L_{min}	L_{max}	S_{min}	S_{max}
set1	8	47	117	59	179	71	88
set2	20	45	75	61	159	68	88
set3	11	116	299	355	8245	64	100
set4	20	13	158	411	120762	67	97
set5	14	10	36	64	10539	56	96
set6	29	49	500	36	759	62	91
set7	12126	10	400	147	152160	37	97
set8	2	144	263	8619	10305	74	87
set9	40	50	200	1287	1287	74	94

Table 5.2 – Properties of the problems of the knowledge base

5.4 Prediction with the best score

In this section we will explain the method that we have designed in order to predict with a good accuracy the best score of a problem but taking into account the instance in the prediction formula. It will prepare a simple foundation on which to extend to a more complex method when the best score of the instance to predict is not known and will serve:

1. to prove if the MLR is a valid model for the prediction
2. to detect which part of the MLR has more influence for the prediction.

3. treebase.org
4. www.treefam.org/
5. Ribosomal RNAs (rRNA) perform critical functions in the ribosome that allow protein synthesis to occur. The genes that encode rRNAs evolve in a very specific manner that makes them excellent markers to trace evolutionary history and powerful tools to identifying species from sequence data.
6. http://www.cs.utexas.edu/users/phylo/datasets/phylogeny-topology.html
7. 16S rDNA are used in reconstructing phylogenies because of the slow rates of evolution of this region of the gene.
8. http://www.microbesonline.org/fasttree/#16S

5.4.1 Problem properties

Let's consider a problem P for which we have the following information:

— K the number of sequences
— L the length of a sequence (all taxa have the same length)

we can compute a first set of properties:

— R the average score (over 100 runs) of initial solutions generated randomly
— N the score of the tree obtained from the neighbor-joining algorithm
— G the average score (over 100 runs) of initial solutions generated with a greedy algorithm
— B the best score found by TNT, SAMPARS or any over parsimony-based software
— S the similarity of sequences computed as the number of equal pairs over the total number of pairs of residues

and a second set of properties (see Figure 5.1):

— RN the decrease in percentage from R to N: $(R - N)/R \times 100$
— RG the decrease in percentage from R to G: $(R - G)/R \times 100$
— RB the decrease in percentage from R to B: $(R - B)/R \times 100$
— NB the decrease in percentage from N to B: $(N - B)/N \times 100$
— NG the decrease in percentage from N to G: $(N - G)/N \times 100$
— GB the decrease in percentage from G to B: $(G - B)/G \times 100$

The scores R, N, G, B are such that $R > N \geq G \geq B$. However in some cases we have $G > N$ (21 % of the database for a total of 2,686 problems).

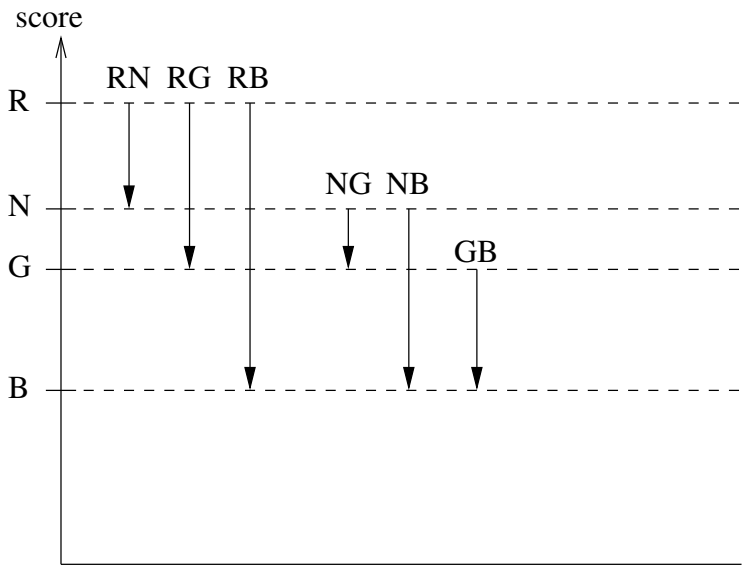


Figure 5.1 – Percentage of decrease between random (R), neighbor-joining (N), greedy(G) and best (B).

On Table 5.3 we provide some information about $set1$ and $set2$ especially for the properties RN, RG. The $set1$ of problems is quite easy to solve, we can reach a value close to the best score very quickly. On the contrary the problems of $set2$ are very difficult to solve. We can note the difference by comparing columns RN and RB:

— for $set1$, RN ranges from 39 to 68 and RB from 43 to 70
— for $set2$, RN ranges from 10 to 21 and RB from 17 to 28

The column RNB which stands for the formula $100 - (RN/RB \times 100)$ indicates if we are close to the optimum B when we start from N knowing R[9]. When this quantity is small it means that the tree generated by neighbor-joining is close to B and it could be considered as one of the measures of the difficulty of a problem. For $set1$ it ranges from 3 to 10 %, and for $set2$ it ranges from 22 to 41 %. Among all problems of the database it concerns 68 instances if we chose $RNB > 10$. Note that we could also use $|RN - RB| = NB$, which is the last column of the table but it seems less obvious to infer the difficulty of an instance with this quantity.

problem	K	L	R	N	G	B	S	RN	RG	RB	RNB	NB
ANGI	49	59	421	234	229	216	77	44	46	49	9	5
CARP	117	110	1520	592	590	548	82	61	61	64	5	3
ETHE	58	86	878	385	388	372	71	56	56	58	3	2
GOLO	77	97	883	535	535	497	84	39	39	44	10	5
GRIS	47	93	470	195	183	172	88	59	61	63	8	4
ROPA	75	82	719	353	347	326	82	51	52	55	7	4
SCHU	113	146	2559	810	803	759	74	68	69	70	3	2
TENU	56	179	1568	728	710	682	76	54	55	57	5	3
tst01	45	61	723	594	586	545	68	18	19	25	28	7
tst02	47	151	1631	1465	1426	1354	75	10	13	17	40	7
tst03	49	111	1067	899	895	833	82	16	16	22	28	6
tst04	50	97	787	663	645	587	88	16	18	25	38	9
tst05	52	75	1015	855	844	789	68	16	17	22	29	6
tst06	54	65	816	666	649	596	75	18	20	27	32	9
tst07	56	143	1590	1401	1364	1269	82	12	14	20	41	8
tst08	57	119	1114	945	929	852	87	15	17	24	35	9
tst09	59	93	1447	1235	1216	1158	68	15	16	20	27	5
tst10	60	71	991	795	787	734	75	20	21	26	24	6
tst11	62	63	771	620	601	552	82	20	22	28	31	8
tst12	64	147	1546	1357	1310	1245	87	12	15	19	37	7
tst13	65	113	1912	1617	1608	1535	68	15	16	20	22	7
tst14	67	99	1524	1266	1252	1174	76	17	18	23	26	6
tst15	69	77	1062	839	834	768	82	21	21	28	24	7
tst16	70	69	776	617	600	555	88	20	23	28	28	8
tst17	71	159	2970	2588	2581	2475	68	13	13	17	23	10
tst18	73	117	1962	1666	1637	1551	76	15	17	21	28	6
tst19	74	95	1382	1124	1112	1041	82	19	20	25	24	6
tst20	75	79	955	769	741	688	88	19	22	28	30	9

Table 5.3 – Data for set1 and set2

5.4.2 The method

The problem to predict will be represented by a tuple $P_0 = (K_0, L_0, ..., GB_0)$. The algorithm is composed of three steps, namely:

1. problems selection
2. multiple linear regression model computation

9. *RNB*: note that RNB is different from NB

3. estimation using the model

5.4.2.1 Problems selection

We select from the database all problems $P = (K, L, ..., GB)$ that are close to P_0 and for which the four following criteria are met:

— $RN = RN_0 \pm \delta$
— $RG = RG_0 \pm \delta$
— $RB = RB_0 \pm \delta$
— $S = S_0 \pm 20$

Initially $\delta = 0.5$ and we retain the first 10 problems that satisfy the constraints and we add P_0 to this set. If the selection does not provide 10 problems we increase δ by a factor of 0.5. In some cases we have a lack of problems close to P_0.

5.4.2.2 Linear regression

We compute lm a multiple linear regression model taking into account the properties K, L, R, N, G, S, RN, RG, RB in order to determine B. For this we use the *GSL* (GNU Scientific Library) and especially the function `gsl_multifit_linear`. We can then determine the average error over all the problems.

5.4.2.3 Estimation

We compute E_0 which is the score estimated from the lm model using the data of P_0.

5.4.2.4 Results and analysis

If we consider a prediction with an accuracy of 1%, i.e. the score we predict E is such that $E = B \pm 1\%$, then we get the following results (see Table 5.4):

— 12,132 instances were predicted successfully, so that means 98.88% of the database
— 135 instances have an accuracy greater than 1%
— 3 problems could not be predicted because we could not get enough close instances to use the MLR.

% of error	within range	outside range	not predicted	error		
				min	max	avg
1%	12,132 (98.88%)	135	3	0	297	8.51
2%	12,231 (99.68%)	36	3	0	297	8.71
3%	12,247 (99.81%)	20	3	0	297	8.75
4%	12,255 (99.88%)	12	3	0	297	8.78
5%	12,259 (99.91%)	8	3	0	297	8.80
10%	12,264 (99.95%)	3	3	0	297	8.81

Table 5.4

If we remove the instance to be predicted from the MLR then we get results of bad quality (see Table 5.5) with only 67.7% of the problems that could be predicted with an error of 1%.

% of error	within range	outside range	not predicted	error		
				min	max	avg
1%	8,307 (67.70%)	3960	3	0	4260	115.83
2%	9,754 (79.49%)	2513	3	0	8755	140.76
3%	10,427 (84.98%)	1840	3	0	8755	155.25
4%	10,804 (88.05%)	1463	3	0	13278	164.84
5%	11,056 (90.11%)	1211	3	0	13278	173.22
10%	11,604 (94.57%)	663	3	0	41082	199.01

Table 5.5

The results obtained prove that the multiple linear regression model is a good candidate for the prediction of the optimum score of the problem but only if we have problems close to the instance to predict and that, in some way, we already know the solution we want to predict when we integrate P_0 into the MLR.

5.5 Prediction without the best score

We follow the same scheme as before but in this case we do not know B_0 so we could not use the selection criterion related to RB. The MLR will concern the properties K, L, N, R, G, S, RN, RG.

5.5.1 Problems selection

5.5.1.1 default selection

We use the same selection criteria as in Section 5.4.2.1 and we select from the database all problems $P = (K, L, ..., GB)$ that are close to P_0. The results are of poorer quality (see Table 5.6)

% of error	within range	outside range	not predicted	error		
				min	max	avg
1%	9377(76.42%)	2885	8	0.00	119.99	4279.31
2%	10963(89.35%)	1299	8	0.00	163.51	7289.59
3%	11444(93.27%)	818	8	0.00	180.93	18043.94
4%	11680(95.19%)	582	8	0.00	191.68	18043.94
5%	11808(96.23%)	454	8	0.00	201.63	18043.94
10%	12037(98.10%)	225	8	0.00	216.49	27572.03

Table 5.6 – Prediction without best score but with same criteria as in section 5.4.2.1

5.5.1.2 Modification of problem selection

In order to get a prediction of better quality we decided to modify the selection criteria, from the database we select instances closer to the problem P_0.

In this case the three following criteria are met:

— $RN = RN_0 \pm \delta$
— $RG = RG_0 \pm \delta$

— $S = S_0 \pm \gamma$

Initially $\delta = 0.5$ and $\gamma = 5$. If the selection does not provide 10 problems we increase δ by a factor of 0.5 and γ by a factor of 5. The results show an increase of the prediction (see Table 5.7).

% of error	within range	outside range	not predicted	error		
				min	max	avg
1%	10627 (86.61%)	1635	8	0	4279	99.81
2%	11543 (94.07%)	719	8	0	7289	134.07
3%	11782 (96.02%)	480	8	0	18043	147.17
4%	11909 (97.06%)	353	8	0	18043	156.69
5%	11985 (97.68%)	277	8	0	18043	164.95
10%	12109 (98.69%)	153	3	0	27572	175.87

Table 5.7 – Prediction without best score but with an increased selectivity

5.5.1.3 Enhanced problem selection

By studying some predictions we could observe that some of the problems selected to compute the MLR model are sometimes not retained in the final list of 10 problems that constitutes the model. For example we could have a selection of 50 instances but we will retain only 10 out of those 50 instances to compute the MLR. In order to have close instances to P_0 we have decided to define a distance from the selected instances to the problem to predict that will be used to order the problems.

If P and P_0 are close in terms of properties then the $RNGS$ distance will be close to 0. This distance is simply the product of the differences of the properties of P and P_0 and was determined empirically. We needed to add some constant so that if two properties are equal then the overall expression would not be 0 even if the other properties differ.

$$\begin{array}{rccccc} rngs(P,P_0) = & (|RN - RN_0| + cst1) & \times & (|RG - RG_0| + cst1) & \times \\ & (|NG - NG_0| + cst1) & \times & (|S - S_0| + cst2) & \times \\ & (|K - K_0| + cst2) & \times & (|L - L_0| + cst2) & \end{array}$$

where $cst1 = 0.001$ and $cst2 = 0.01$.

We then sort the problems according to their increasing $RNGS$ distance. The $RNGS$ distance needs probably to be refined. For example the range of the length of the taxa L is important (from 36 to 152160), it is obvious that the influence of $|L - L_0|$ can sometimes be too important. Maybe it will be more interesting to use a logarithm to decrease the influence of L. We need to study the influence of each parameter in order to determine a formula with a better precision.

5.5.2 Estimation by weighted average (WA)

In order to improve the quality of the prediction we have decided to use a method different from the MLR. As we have the RB, NB and GB values of the selected problems we have decided to use them in order to predict the optimal score of P_0

We simply compute an average of the scores predicted using RB, NB, GB. Given 10 instances close to P_0 we compute:

$$predicted_{RB} = \frac{1}{10}\sum_{i=1}^{10} R_0 - \frac{(1 - RB_i)}{100}$$

$$predicted_{NB} = \frac{1}{10}\sum_{i=1}^{10} N_0 - \frac{(1 - NB_i)}{100}$$

$$predicted_{GB} = \frac{1}{10}\sum_{i=1}^{10} G_0 - \frac{(1 - GB_i)}{100}$$

For example, $predicted_{RB}$ is the average of the predictions using the RB property. The final score predicted is given by the weighted average:

$$predicted = \frac{2 \times predicted_{GB} + 2 \times predicted_{NB} + predicted_{RB}}{5}$$

Here it is obvious that GB and NB being closer to B will have less inaccuracy than RB. So we increase the weight of GB and NB.

5.5.3 Results

Table 5.8 shows the predictions obtained by this new method using:

1. the enhanced selection using the $RNGS$ distance to order the selected problems
2. the weighted average to compute the prediction of the score

With this new method we get results of very good quality with 96.23% of the instances predicted at $\pm$ 1% from the best score.

However, with an accuracy of 5% the problems not predicted are the following:

```
CARP GOLO GRIS tst01 tst02 tst04 tst05 tst06 tst07
tst08 tst10 tst11 tst12 tst14 tst15 tst16 tst19
tst20 m0972 astr bracon dinos kearney norell
```

Those problems are predicted with a precision between 5% and 9%. This is quite strange and needs further investigation because the problems `tst01` to `tst20` problems formed the basis of the predictor at first. We could observe that there are many TF problems (from *set7*) which appear as selected and retained to establish the prediction.

% of error	within range	outside range	not predicted	error		
				min	max	avg
1%	11808 (96.23%)	454	8	0.00	90.81	4579.12
2%	12194 (99.38%)	68	8	0.00	96.74	4679.34
3%	12221 (99.60%)	41	8	0.00	96.94	4679.34
4%	12229 (99.67%)	33	8	0.00	96.91	4679.34
5%	12238 (99.74%)	24	8	0.00	96.85	4679.34
10%	12262 (99.93%)	0	8	0.00	96.76	4679.34

Table 5.8

5.5.3.1 Example that works: HIV

Here is a detailed output of the prediction for instance HIV of best score $73,537$. We could predict a score of $73,500.17$ which corresponds to an error of 36.83, i.e. 0.05% of the score. The closest instances selected are at least 207. They were sorted according to the $RNGS$ distance and only the 10 closest to HIV were kept.

```
for HIV, K=94.00, L=10922.00 RN=41.83, RG=41.83, S=79.00
selection criterion RN : [41.33..42.33]
selection criterion RG : [41.33..42.33]
selection criterion S  : [69.00..89.00]
instances=207
....
selected
problem;   K ;    L ;     R ;     N ;     G ;     B ; S ;     RB ;     NB ;     GB ;   RNGS
-------------------------------------------------------------------------------------------
TF105441;  90; 17727; 213026; 124092; 124010; 122700; 79; 42.4014;  1.1217;  1.0564; 426607
TF313930;  94;  3081;  22223;  12910;  12901;  12770; 87; 42.5370;  1.0844;  1.0154; 2515803
TF105857;  94;  5523;  61343;  35473;  35440;  35042; 82; 42.8753;  1.2150;  1.1230; 17847250
TF312937;  99;  6462;  87629;  51257;  51184;  50542; 79; 42.3227;  1.3949;  1.2543; 34305572
TF315769; 156;  3741;  80695;  47049;  46977;  46577; 79; 42.2802;  1.0032;  0.8515; 66976884
TF314922; 110;  3918;  56416;  32561;  32538;  32219; 79; 42.8903;  1.0503;  0.9804; 114962584
TF324966;  95;  6045;  53111;  30990;  30977;  30600; 85; 42.3848;  1.2585;  1.2170; 128601128
TF105725; 103;  8424; 117288;  68711;  68187;  67603; 79; 42.3615;  1.6126;  0.8565; 172085920
TF324756;  90;  2073;  28684;  16724;  16728;  16564; 76; 42.2535;  0.9567;  0.9804; 184317104
TF103043; 107;   702;  11331;   6613;   6612;   6486; 76; 42.7588;  1.9205;  1.9056; 211237824

residual error=0.0146
r^2=1.0000
-1984.4419 + 0.0844 * K - 0.0842 * L + 0.1380 * R + 4.4305 * N
- 3.6679 * G + 28.3304 * S + 5037.9605 * RN - 5042.8250 * RG

    TF105441  122700.00 =>  122697.93 +/- 2.07
    TF313930   12770.00 =>   12765.15 +/- 4.85
    TF105857   35042.00 =>   35041.90 +/- 0.10
    TF312937   50542.00 =>   50543.26 +/- 1.26
    TF315769   46577.00 =>   46577.08 +/- 0.08
    TF314922   32219.00 =>   32221.41 +/- 2.41
    TF324966   30600.00 =>   30605.42 +/- 5.42
    TF105725   67603.00 =>   67603.01 +/- 0.01
    TF324756   16564.00 =>   16566.08 +/- 2.08
    TF103043    6486.00 =>    6481.74 +/- 4.26
min error=      0.01
max error=      5.42
average error=      2.25
predicted_rb=  73523.18
predicted_nb=  73449.38
predicted_gb=  73548.87
score_wa    =  73500.17
score_mlr   =  73510.84
score=73500.17, error=36.83, percentage=0.05
```

We have included both methods:

— the multiple linear regression with a $R^2 = 1.000$ (see variable `score_mlr`)
— the weighted average (see variable `score_wa`)

It appears that in this case both methods have a high accuracy but the MLR gives the best prediction. However we use the method based on weighted average to provide the result. The WA works fine because for HIV, we have:

— $RB_0 = 42.50$ and the average of RB for the selected instances is 42.51
— $NB_0 = 1.14$ and the average of NB for the selected instances is 1.26
— $GB_0 = 1.14$ and the average of GB for the selected instances is 1.12

5.5.3.2 Example that does not work: tst07

In this case both prediction methods (MLR and WA) provide results far from the optimum of 1269. Here we can observe that the values of RB, NB and GB differ for `tst07` and the selected instances:

— $RB_0 = 20.19$ and the average of RB for the selected instances is 14.73
— $NB_0 = 9.42$ and the average of NB for the selected instances is 2.58
— $GB_0 = 6.96$ and the average of GB for the selected instances is 1.41

```
for tst07, K=56.00, L=143.00 RN=11.89, RG=14.21, S=82.00
selection criterion RN : [11.39..12.39]
selection criterion RG : [13.71..14.71]
selection criterion S  : [72.00..92.00]
selection criterion RN : [10.89..12.89]
selection criterion RG : [13.21..15.21]
selection criterion S  : [67.00..97.00]
selection criterion RN : [10.39..13.39]
selection criterion RG : [12.71..15.71]
selection criterion S  : [62.00..102.00]
instances=19
...
keep=10

problem ; K ;    L ;    R ;    N ;    G ;    B ; S ;      RB ;     NB ;     GB ;  RNGS
-------------------------------------------------------------------------------------------------------------------
    mh3; 20;    64;   139;   122;   119;   118; 80;  15.1079;  3.2787;  0.8403; 1158066304
  tst12; 64;   147;  1546;  1357;  1310;  1245; 87;  19.4696;  8.2535;  4.9618; 3897193728
TF353269; 11;   627;   764;   669;   657;   657; 79;  14.0052;  1.7937;  0.0000; 542693228544
  tst17; 71;   159;  2970;  2588;  2581;  2475; 68;  16.6667;  4.3663;  4.1069; 2064919298048
TF322948; 23;  5967; 17288; 15240; 15083; 14931; 79;  13.6337;  2.0276;  1.0078; 9093992415232
TF351966; 13;   339;   577;   501;   498;   494; 77;  14.3847;  1.3972;  0.8032; 11864166629376
TF339643; 23;  2061;  4084;  3582;  3553;  3534; 84;  13.4672;  1.3400;  0.5348; 21044804976640
TF341912; 11;   567;  1163;  1028;  1015;  1011; 68;  13.0696;  1.6537;  0.3941; 21141571764224
TF336379; 18;  1098;  2144;  1865;  1862;  1849; 81;  13.7593;  0.8579;  0.6982; 27015384137728
TF340439; 18;   414;  1051;   915;   914;   907; 75;  13.7012;  0.8743;  0.7659; 57604751491072

residual error=0.0346
r^2=1.0000
73.0874 - 0.7553 * K + 0.0059 * L - 0.8703 * R + 0.5607 * N
+ 1.4168 * G + 2.7111 * S - 2.5416 * RN - 16.8288 * RG

mh3     118.00 =>     118.07 +/- 0.07
      tst12    1245.00 =>    1245.05 +/- 0.05
    TF353269     657.00 =>     656.49 +/- 0.51
      tst17    2475.00 =>    2474.97 +/- 0.03
    TF322948   14931.00 =>   14930.99 +/- 0.01
    TF351966     494.00 =>     494.51 +/- 0.51
    TF339643    3534.00 =>    3534.01 +/- 0.01
    TF341912    1011.00 =>    1011.24 +/- 0.24
    TF336379    1849.00 =>    1849.24 +/- 0.24
    TF340439     907.00 =>     906.43 +/- 0.57
min error=      0.01
max error=      0.57
average error=     0.22
predicted_rb=   1355.85
predicted_nb=   1364.79
predicted_gb=   1344.75
score_wa    =   1354.82
score_mlr   =   1318.95
score=1354.82, error=85.82, percentage=6.76
```

5.6 Conclusion

In this chapter we have tried to design the key components of a predictor for the Maximum Parsimony problem based on a knowledge base of solved instances and on a multiple linear regression model. We observed that the MLR was interesting only in the case when we knew what we had to predict. If we removed the RB property from the MLR the percentage of the prediction falls below 86% with the modified problem selection.

In order to get an improved accuracy we had to substitute the MLR by a weighted average which proved to give very interesting results with a percentage of prediction close to 96%. Given that we have all data of the instances, we can deduce that the predictor does not work if the average of RB, NB and GB of the selected instances used for the prediction are far from the properties RB_0, NB_0 and GB_0 of the instance to predict.

In the general case, for an instance that does not belong to the database we do not know the values of the RB_0, NB_0 and GB_0 properties as B_0 is unknown.

However, the method can provide the user with some useful information that can be harnessed to determine if the instance will be difficult to solve or not and consequently use the appropriate set of parameters for a software. We need to find more accurate measures to estimate the difficulty of the problem as for example the *zilla* instance (500 taxa, 759 residues) has a RNB

of 1.8 which makes it a problem easy to solve while it is not the case because the best optimal score known of 16218 is difficult to reach. However it is not too difficult to reach a score of 16221 or 16220 for example by using SAMPARS. The difficulty of *zilla* probably comes from the important number of taxa. We also need to refine the method to analyze the cases that lead to a bad prediction. This study nonetheless is very encouraging and is spurring us on to greater efforts.

6

Improvement of the evaluation of the objective function

6.1 Introduction

The objective or fitness function is one of the key elements for the successful implementation of metaheuristic algorithms because it is in charge of guiding the search process toward good solutions in a combinatorial search space. Previous works for solving the MP problem have commonly evaluated the quality of a potential solution using the parsimony cost depicted in equation (2.5) (Andreatta and Ribeiro, 2002; Barker, 2003; Ribeiro and Vianna, 2005, 2009; Richer, Goëffon, and Hao, 2009). In terms of performance the objective function is called every time a new solution must be evaluated and should have the shortest execution time.

In this chapter we introduce two implementations to compute the objective function of MP, the first one is the conventional way to compute it using the CPU (processor) and the second one is a GPU version. This objective function is the most time consuming (Bader, Chandu, and Yan, 2006) but exhibits much parallelism. Generally GPUs are used to perform floating point calculations but in this case we will only use integers and take advantage of the many cores / threads of the GPU.

6.2 Background

The evaluation function is used during different phases:

— when building an initial solution in a greedy manner, the objective function is used to evaluate the addition of a taxon x to any node z of a subtree in order to minimize the increase in parsimony cost $\phi(z)$ (see 2.3); during this step, the algorithm counts the number of unions (or mutations) of the new hypothetical taxon,
— when we generate a new neighbor using NNI, SPR or TBR, a subtree is *degraphed* and *regraphed* on another node of the initial tree, it is then necessary to recompute the score of the new tree obtained.

The techniques that we will talk about in this chapter concern what is called *multi-character* optimization techniques.

A set of methods (Goloboff, 1993; Gladstein, 1997; Ronquist, 2000; Yan and Bader, 2003) fall into the category of *fast character* optimization techniques, i.e. a set of shortcuts that helps decrease the computation time by not recalculating the whole tree each time a SPR or TBR modification is applied. Those techniques are particularly effective when an important number of SPR or TBR neighbors has to be evaluated. In the case of Fitch's parsimony, characters are considered as unordered and

multistate: they can transform from one state to another independently. As a consequence, an unrooted tree may be rooted on any branch with no modification of the parsimony score, which means there is a potential root node for any branch.

In (Goloboff, 1993), the author proposed a method for indirect calculation of the parsimony score which uses two passes. This method needs only to compare the root of the clipped tree with the potential root of the target tree to obtain the score of a potential new tree for a SPR search. Gladstein (1997) also proposed an algorithm which is exact and correct. In (Yan and Bader, 2003) a two passes algorithm is described which has the same complexity of Goloboff's and is faster than the incremental method of Gladstein.

However those *fast character* optimization techniques are difficult to implement and are effective with basic C implementation.

6.3 CPU implementation

In this section we briefly describe some implementation details to efficiently compute the parsimony score $\phi(t)$ of a tree using the SIMD (Single Instruction on Multiple Data) units of the CPU (Central Processing Unit). This version will serve as a reference for our benchmarks. The most time consuming function in a search algorithm for the resolution of MP in our case is the function of Fitch (see Algorithm 13).

Algorithm 13: Fitch algorithm to compute hypothetical sequence z from two sequences x and y and evaluate the number of mutations

input: x, y: array[k] of sets of characters
output: z: array[k] of sets of characters, $mutations$: number of mutations

```
1 mutations ← 0;
2 i ← 0;
3 while i < k do
4     z[i] ← x[i] ∩ y[i] ;
5     if z[i] = ∅ then
6         mutations ← mutations + 1;
7         z[i] ← x[i] ∪ y[i];
8     i ← i + 1 ;
9 return mutations ;
```

This function will be called to compute all the hypothetical sequences z for each node of the tree. The sum of all mutations will constitute the parsimony score of a tree.

In order to efficiently perform the union and intersection of Algorithm 13, each character is represented by a power of 2. For nucleotides the different symbols are (`-`, `A`, `C`, `G`, `T`, `?`). We represent the gap symbol `-` by $2^0 = 1$, until `T` $2^4 = 16$. The undefined character `?` which can represent any other character is then coded by the value $31 = 1 + 2 + \cdots + 16$. With this representation the union can be performed by the binary-OR (`|`) and the intersection by the binary-AND (`&`). This gives rise to the C version of the previous algorithm (see Figure 6.1).

Although quite simple this function can not be vectorized by C++ compilers like GNU g++ or Intel icpc. Nevertheless, we can implement an efficient version of the Fitch function in assembly by taking full advantage of some relevant features offered by modern x86 processors. More precisely, the core of modern x86 processors has a SSE (SIMD Streaming Extension) or AVX (Advanced Vector Extensions) unit which enables to treat data as *vectors*. The use of vectors let us perform the same operation on different data at the same time. Recent Intel processors offer on a 32-bits architecture a set of 8 SSE registers of 128 bits or 8 AVX registers of 256 bits long. If we represent a nucleotide with one byte, in the case of DNA, then a SSE register can store and handle 16 bytes (nucleotides) at a time (resp. 32 with AVX). In the case of proteins it would be necessary to use 32 bits integers to represent the 20 different amino acids, so a SSE register would handle 4 integers (resp. 8 with AVX).

The experiments that were carried out by Richer (2008) show that the vectorization of Fitch's function gives a 90% speedup on Intel Core 2 Duo processors compared to the basic version of Figure 6.1, while other architectures (Pentium II/III/4, Pentium-M, Athlon 64, Sempron) provide 70 to 80% improvement. This improvement enables to divide the overall computation time of

```
1  int fitch(char *x, char *y, char *z, int k) {
2    int mutations = 0;
3    for (int i=0; i<k; ++i) {
4      if ((z[i] = x[i] & y[i]) == 0) {
5        z[i] = x[i] | y[i];
6        ++mutations;
7      }
8    }
9    return mutations;
10 }
```

Figure 6.1 – C code of Fitch function.

Table 6.1 – Results in seconds for different implementations and number of threads for 64 bits architecture on an Intel i5-4570 CPU @ 3.20GHz

implementation	threads	time (s)	improvement
c language	1	311	–
c language	4	128	58%
sse4.2	1	10	96%
sse4.2	4	4	98%

a program by a factor of 3 to 4. A first pseudo-code was given in (Ronquist, 1998) for PowerPC processors.

Finally, note that the most recent processors (Intel Core i5 or i7, AMD Phenom) introduce the SSE4.2 instructions set that contains the *popcnt* instruction which counts the number of bits set to one in a general purpose register. This instruction is used essentially to determine the number of mutations that occur when we perform the union between two SSE registers. By replacing a C implementation of *popcnt* by the native SSE4.2 assembly instruction, the experiments carried out show an overall improvement of 95% (on an Intel Core i7 860 processor) compared to the basic implementation. The introduction of the AVX and then AVX2 instructions sets in 2013 by Intel enables to use vectors of 256 bits. Rewriting the SSE4.2 version into AVX2 could only lead of a gain of 1% to 3% compared to the SSE4.2 version on Intel Haswell processors. For more details please see (Richer, 2013).

Some recent tests we made on a descent algorithm using the SPR neighborhood on a new implementation of trees based on an array representation gives the following results (see Table 6.1): by using SSE4.2 we can decrease the execution time by 96% compared to the C language implementation.

6.4 GPU programming

GPU programming also called GP (for General Purpose) GPU programming consists in using a graphics card to perform very intensive and parallel computations. The GPU (Graphics Processing Unit) can be viewed as a co-processor although this analogy is far from reality because the GPU can also be considered like a computer inside the computer as it is formed of cores and memory.

NVidia which sells GPUs since 1993 has developed CUDA [1] a parallel computing platform and application programming interface (API). The CUDA platform is designed to work with programming languages such as C, C++ and Fortran [2] and enables the programmer to write code that will be executed on the GPU, but that will be called from a program executed on the CPU. The interesting point is that we do not have to learn a new language because the code of the GPU, called a *kernel*, can be written

1. Common Unified Device Architecture: http://www.nvidia.com/object/cuda_home_new.html
2. but can also be used in Python or Java for example

in C or C++ for example.

On the point of view of NVidia the CPU is considered as a multi-core device (with 2 to 20 cores [3]) designed for general purpose treatments while the GPU is a many-core device (with hundreds or thousands of cores [4]) designed for massively parallel treatments. However the extra cost of memory transfer from central memory (DDR) to the memory of the GPU (GDR) can sometimes lower down or annihilate the speed-up gained by the many cores.

A good introduction to learn GPU computing with CUDA is the book of Sanders and Kandrot (2010).

6.4.1 Programming model

We use the NVidia CUDA programming model (see Figure 6.2) where:

1. the data stored in the global memory (called DDR) which are processed by the CPU are copied to the RAM of the GPU (called GDR); the kernel is also copied to the GPU,
2. the data in GDR are treated by a kernel executed as many threads dispatched on the GPU cores and the data in the GDR memory are generally modified by the threads,
3. the data in the GDR is then copied back to the DDR to be eventually processed by the CPU.

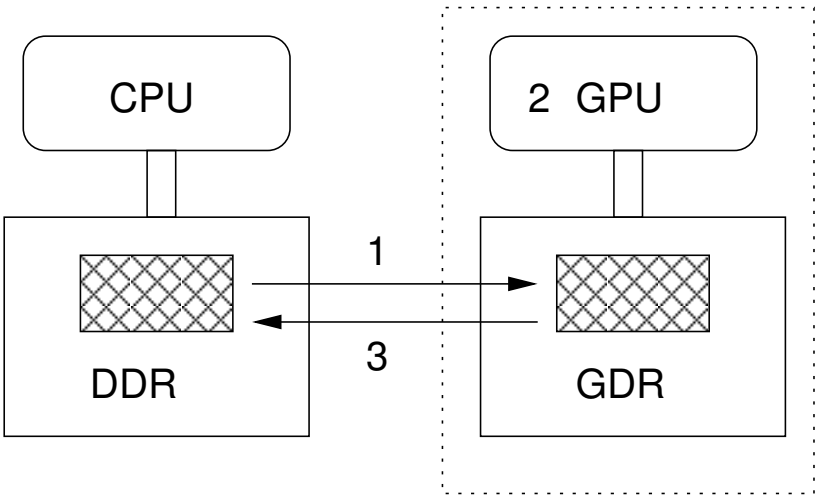


Figure 6.2 – NVidia programming model

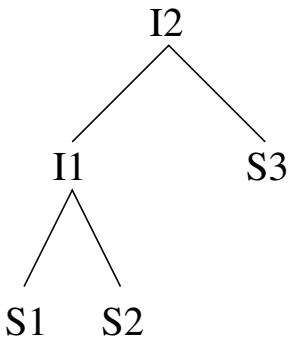
6.4.2 GPU implementation

We now come to the presentation of the GPU implementation of $\phi(t)$. We use a static or flat representation of the tree to speed up the treatments and to obtain coalesced [5] memory transaction when all of the threads in a warp [6] or half-warp access global memory at the same time.

A node is represented by a structure of four integers, namely: L, R, P, N which are the index of the Left and Right offspring and the index of the Parent. The N field is used only for data alignment in memory but also contains the index of the node (see Figure 6.3).

The first n (here $n = 3$) nodes contain the leaves (for example: `S1`, `S2`, `S3`) that have no offspring so the fields L and R are set to -1. The rest of the $n - 1$ nodes are internal nodes. The root node (here `I2`) has no parent so the field P is set to -1. The

3. for example the Intel Xeon E5 2670 of taurus, the cluster of the LERIA, has 10 cores with Hyper-Threading.
4. for example the NVidia GTX Titan X with 3072 CUDA cores.
5. *coalesced*: a NVidia term for contiguous.
6. *warp*: a group of 32 threads which is the minimum size of the data processed in SIMD fashion by a CUDA multiprocessor.



	L	R	P	N
S1	−1	−1	3	0
S2	−1	−1	3	1
S3	−1	−1	4	2
I1	0	1	4	3
I2	3	2	−1	4

Figure 6.3 – Matrix representation of a binary tree (flat tree) with three leaves and two internal nodes

two basic operations on the tree which are the *degraph* (cut of a subtree) and the *regraph* (paste of a subtree on a branch) must be implemented carefully in order to avoid a false interpretation of the score. For example if we put `I2` before `I1` on the table of Figure 6.3, the result will be wrong because `I1` appears before `I2` in the tree when we go from the leaves to the root.

We store the initial and the hypothetical sequences in a matrix M of $(2n-1) \times k$ characters or integers called $data$ on the GPU. The first n rows will not be modified as they contain the initial sequences (OTUs), the other rows will be modified by the kernel to compute the hypothetical sequences or HTUs (see Figure 6.4).

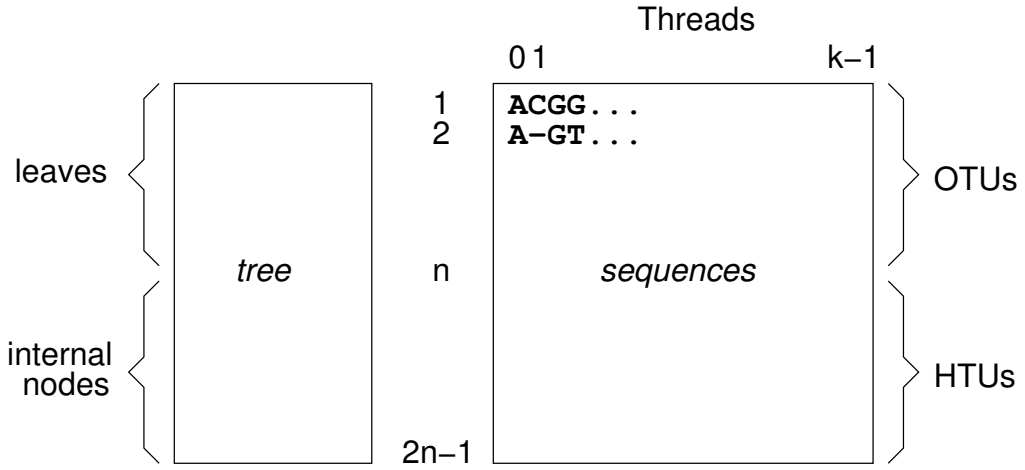


Figure 6.4 – Data representation with sequences in one array and flat tree

The code of the kernel consists in computing each internal sequence by assigning to each thread a column of the matrix M. We start from the first internal sequence at index n and then we move back towards the root (see Figure 6.5). The input parameters are `N` $(= n)$ the number of leaves of the tree, `K` $(= k)$ the length of the sequences and $data$ (previously described). The output parameter is an array of k integers called $mutations$ which will record the number of mutations for each column of the sequences. This array is stored on the GPU. The flat tree is obtained from constant memory (variable gpu_tree) for efficiency reason as the tree is a read-only structure.

The call to the kernel is presented on Figure 6.6. First, the tree to evaluate called cpu_tree is copied to the constant memory area on the GPU gpu_tree. After the execution of the kernel, the mutations are copied from the GPU to the CPU and are then summed on the CPU to obtain the parsimony score $\phi(t)$.

```
__global__ void kernel(int N, int K, Character *data,
    int *mutations) {
  int tid = blockDim.x * blockIdx.x + threadIdx.x;
  int local_mutation = 0;
  int node_id = N;

  while (gpu_tree[node_id].P != -1) {
    int left_node_id = gpu_tree[node_id].L;
    int right_node_id = gpu_tree[node_id].R;
    Character x_i = data[left_node_id * K + tid];
    Character y_i = data[right_node_id * K + tid];
    Character z_i = x_i & y_i;
    if (z_i == 0) {
      ++local_mutation;
      z_i = x_i | y_i;
    }
    data[node_id * K + tid] = z_i;
    ++node_id;
  }

  if (tid < K) {
    mutations[tid] = local_mutation;
  } else {
    mutations[tid] = 0;
  }
}
```

Figure 6.5 – Kernel to compute parsimony score.

```
cudaMemcpyToSymbol("gpu_tree", cpu_tree, (2*N-1) * sizeof(Node),
  0, cudaMemcpyHostToDevice);
kernel<<<grid,block>>>(N, K, gpu_data, gpu_mutations);
Memcpy(cpu_mutations, gpu_mutations, sizeof(int) * K,
  cudaMemcpyDeviceToHost));
int cost = accumulate(&cpu_mutations[0], &cpu_mutations[K], 0);

```

Figure 6.6 – Kernel to compute parsimony score.

6.5 Results

We have implemented a simple benchmark in order to compare the CPU and GPU implementations. The program takes as input the number of sequences and the number of residues of the sequences. The sequences are randomly generated. We then randomly generate 50 different trees and perform 1000 evaluations of the parsimony score for each tree. Results were obtained on the latest Intel CPU architecture (Haswell - Core i5-4570 running at 3.20 GHz with AVX2), a common GPU (GTX 770) and a high end GPU (Tesla K20).

From Table 6.2 we can see that depending of the number of sequences the GPU will become more efficient as long as the number of sequences increases. For 8 bits data, when $n = 64$, the GTX 770 is more efficient than the Core i5 when k is greater than 2 millions residues. When $n = 512$ then it is also the case but with $k \simeq 32768$. The Tesla K20 seems particularly efficient

for large instances. This is also the case with the Tesla K40 (results not reported here). Indeed the Tesla K40 offers 40 percent higher performance than its predecessor, the Tesla K20 and has a boost technology analogous to the Turbo Boost seen in Intel processors. We can also note that it takes more time for Tesla K20 to compute the parsimony score than on the GTX 770 for small lengths ($n = 64, k \simeq 1024$). This is probably due to the *occupancy* factor which is small for small lengths. The occupancy is a metric for quantifying the ability of a given kernel to supply enough parallel work to achieve reasonable performance.

Table 6.2 – Results in seconds for different number of taxa (n) and lengths (k) for 64 bits architecture

taxa (n)	length (k)	i5 4570	GTX 770	Tesla K20	speed up
8 bits data - DNA, RNA					
64	1024	0.03	1.43	1.89	-
	32768	1.59	5.41	5.07	-
	100000	12.50	15.01	11.81	1.05
	200000	36.80	28.90	22.45	1.63
512	1024	0.42	6.14	9.56	-
	32768	38.70	26.36	26.76	1.44
	100000	125.26	82.50	68.49	1.82
	200000	252.39	161.75	131.77	1.91
32 bits data, PROTEINS					
64	1024	0.17	1.43	1.86	-
	32768	17.48	6.78	6.71	-
	100000	58.83	20.33	18.10	3.25
	200000	120.76	39.89	35.35	3.41
512	1024	1.69	6.55	11.05	-
	32768	162.60	38.23	43.09	3.77
	100000	508.90	127.55	122.06	4.15
	200000	1022.30	261.94	243.17	4.2

6.6 Conclusion

In this chapter we have presented the implementation details of the evaluation of the parsimony score of a tree on a CPU and a GPU architectures. The results obtained show that for a small number of sequences of short length the CPU is faster than the GPU. But for an important number of sequences with a long length the GPU becomes much faster than the CPU. This will be very interesting for phylogenies based on multi-genes or whole genomes where sequences can have from thousands to millions of residues.

Another area of research would be to evaluate many regraphs at the same time on the GPU on the same principle of (Robilliard, Marion-Poty, and F., 2008): instead of having a parallel version of the objective function we could have a parallel version of the SPR or TBR neighborhoods on the GPU. This work needs some amount of time to study the feasibility of the solution, design the algorithm and implement the code.

Parallelism can also be used with multi-core CPUs for example when we use the NNI, SPR or TBR neighborhoods using OpenMP[7]. It enables to evaluate different regraphs of subtrees in parallel.

7. This was a project of students of the Master 2 Professional of the Faculty of Science of Angers in 2015.

7

Conclusion and perspectives

7.1 Analysis of the work

In this thesis we have tried to address the Maximum Parsimony problem by developing new methods and ideas to solve it:

- We have designed a new Simulated Annealing algorithm much more interesting and accurate than the LVB implementation: the resolution time is not erratic like in the case of LVB and we provide solutions of better quality. Although relatively slow (compared to TNT) the software SAMPARS could benefit from an implementation close to the one of TNT, but the source code of TNT is not available. Recently, Pablo Goloboff the author of TNT has released Oblong (Goloboff, 2014) which source code is available and is close to the one of TNT. We are currently designing a new implementation to handle trees in a static way which should be more efficient than the present implementation and closer to the one of Oblong.
- From the experiments we have carried out the Path-Relinking approach does not seem to generate improvements most of the time. However the bottom-up implementation we have designed could be used as a measure of distance between trees. It is used in the Arbalet [1] Java software than can compare trees given in newick format. We now offer the possibility to the user to print the tree obtained by SAMPARS in newick format or to visualize it calling Arbalet.
- The prediction of the score of some instance is a very interesting tool that enables the experimenter or the software designer to decide what kind of parameters he has to use in order to reach a good solution in a reasonable time based on the measure of difficulty of the problem. Much work remains to be done to sharpen the accuracy of the prediction in particular for the instances that are predicted incorrectly at this time.
- The GPU implementation could be used to help decrease the computation time for large instances and could be selected instead of an assembly function. The GPU integration needs to be finalized. An interesting alternative to tackle the problem of long sequences is Oblong (Goloboff, 2014) which was designed for parsimony analysis of long sequences storing the data into the disk as blocks. However as pointed out by the author, using disk files to store the data slows down searches by a factor 4 to 5 times.

Our ideas have been integrated into the **biosbl** library [2], a set of classes and programs used to handle input/outputs of bioinformatics data files and basic operations as multiple alignment and phylogenetic reconstruction with distance methods and Maximum Parsimony (see Appendix A).

The Maximum Parsimony problem is a very interesting optimization problem. What makes it all the more interesting, but also difficult to solve is the fact that the solutions of the problem are not vectors of values but trees. The static representation

1. www.info.univ-angers.fr/pub/richer/ur.php?arbalet
2. https://sourceforge.net/projects/biosbl/

of trees is supposed to decrease the computation time as the tree can reside in the L2 cache of the processor and that the data are contiguous in memory. However the dynamic representation with pointers to left and right sons for binary trees is easier to handle. A trade-off between efficiency and ease of use must be found. All those implementation details, although important, were not taking into account as we have decided to write a generic code to test our ideas, especially to represent trees and to write the metaheuristic algorithms.

7.2 Further reflection

7.2.1 Reduction of the problem

Another approach, not developed here, and that could be very useful is a kind of *multi-level* or reduction resolution. This idea is not implemented in TNT but TNT uses the notion of cluster, where a cluster is a subtree of close taxa. TNT tries to optimize clusters, generally at the end of the search, but it would also be interesting to start the search by replacing a cluster by a unique sequence which is the taxa of the root of the cluster and solve the modified version of the problem during some iterations. As the number of taxa has an important influence on the resolution time it could enable us to decrease the overall computation time. When we reach some local optimum we can decide to come back to the original version of the problem by replacing the root sequence by the subtree that represents the cluster.

7.2.2 Operator to escape from local optimum

The methods used to escape a local optimum are the following:

— use a different neighborhood,
— change the objective function (like the Ratchet technique),
— perturb the current solution,
— start the search from another initial solution.

However the perturbation of a solution is sometimes too simple and after a few moves we come back to the previous local optimum. It would be interesting to detect nodes that are potentially cause of a high contribution to the score of the tree and that could be degraphed and processed to build a new tree far from the last local optimum.

7.2.3 Predictor

The predictor could be integrated to many software or as a web service in order to help determine which set of parameters is suitable for the resolution of a particular instance. For the moment the database of instances is integrated in the code of biosbl library as a C array. It would be interesting to make it a database available by Internet and constitute a set of hard instances that could help test the efficiency and accuracy of Maximum Parsimony solvers.

Another interesting study would be to determine if the RN, RG and NG values computed for all instances are correlated. In other words, knowing RN, RG and NG can we infer the values of RB, NB and GB. If it is the case we could increase the accuracy of the predictor.

7.2.4 Multi-agent resolution

As the biosbl package now integrates the work of Goëffon (2006); Goëffon et al. (2008) and my own work it could be possible to start in parallel the resolution of a problem using different metaheuristics (Simulated Annealing, Memetic Algorithm, Iterated Local Search, ...) and let the programs cooperate and exchange information during the search. For example when a new improving solution is found it could be broadcasted to the other software and they could take into account this solution in the next iteration of the search.

Software

A.1 Introduction

The ideas and concepts we have developed have been integrated in the **biosbl** library [1], a set of classes used to handle input/outputs of bioinformatics data files and basic operations like multiple alignment and phylogenetic reconstruction with distance methods and maximum parsimony.

We have designed two tree implementations: a dynamic one that is based on pointers and a static one based on an array representation. Those implementations give rise to different versions of a software. It should probably be more clever and less troublesome for the user to design a unique version which hides the implementation details and to allow the user to chose the implementation using command line parameters. However this is the choice of the last version of the implementation (version 3.1).

The different problems used to create the database are under the `benchmarks/phylo` directory. The files have been stored under different formats: fasta, phylip, nexus or *sequence group*.

A.2 Programs

A.2.1 Sampars

SAMPARS, which stands for *Simulated Annealing for Maximum PARSimony* is the main software that helps solve the Maximum Parsimony problem. There are two versions following the tree implementation: `sampar_dynamic.exe` and `sampar_static.exe`.

The set of command line parameters can be obtained using the `-h` or `--help` option:

```
NAME
sampars_dynamic.exe - Simulated Annealing for Maximum
Parsimony

SYNOPSIS
sampars_dynamic.exe [OPTIONS] ...

DESCRIPTION

--alphabet=VALUE or -a VALUE
  where value=na, aa
```

1. https://sourceforge.net/projects/biosbl/

```
alphabet of input taxa file, use 'na' for nucleic acid and
'aa' for amino acid

--cooling-factor=FLOAT within [0.96..0.9999]
cooling factor (alpha) of simulated annealing, default value
is 0.99

--descent-type=VALUE
  where value=strict, loose
type of descent

--effort=VALUE
  where value=small, medium, large
will determine the number of iterations of the markov chain
(see also --max-mc-iterations)

--final-temperature=FLOAT within [1e-06..3.8]
final temperature of simulated annealing, default value is
1e-5

--heating-factor=FLOAT within [1.00001..1.8]
reheat factor (beta) of simulated annealing, default value
is 1.4

--help or -h
help flag, print synopsis

--improvement-frequency=NATURAL
number of step in the Markov chain after which a descent is
performed to improve neighbor

--initial-solution=VALUE
  where value=random, nj, greedy
method used to generate initial solution

--initial-temperature=FLOAT within [1..7]
initial temperature of simulated annealing, default 0 means
it will be evaluated

--input-file=STRING or -i STRING
input file (same as --taxa-file), use @file_name to look for
the file in the current directory and subdirectories

--markov-chain-length=NATURAL
maximum number of Markov chain iterations (see also
--effort)

--max-reheat=NATURAL
maximum number of times a reheat is performed

--max-threads=NATURAL
maximum number of threads used by OpenMP, use this parameter
if you are using the 'spr_omp' neighborhood

--neighborhood=VALUE
  where value=spr, spr_1, spr_omp, spr_pr, tbr
neighborhood used by the local search algorithm

--output=STRING or -o STRING
output file, used to redirect output

--output-newick=STRING or -n STRING
print final tree into newick to file

--random-generator-seed=NATURAL
set random number generator seed

--silent
silent mode

--stuck-limit=NATURAL
number of iterations that causes a reheat if no improvement
is found

--taxa-file=STRING
input file (same as --input-file), use @file_name to look
```

```
for the file in the current directory and subdirectories

--verbose-level=NATURAL or -v NATURAL
verbose level, chose between 0 (silent) and 5, default is 1

--view=VALUE
  where value=text, arbalet
type of visualization
```

For example to solve problem `tst01` of $set2$, we can use:

```
build/obj/v3.1/release/gnu/programs/sampars_static.exe
  --input-file=benchmarks/phylo/set2/tst01.fasta
```

We can use the script `bin/links.sh` that will create symbolic links to the binaries. So we can after calling the script, write:

```
./sampars_static.exe --input-file=benchmarks/phylo/set2/tst01.fasta
```

If we need to solve a hard instance we could use the TBR neighborhood with an important Markov Chain Length and using an initial solution generated by a greedy process:

```
./sampars_static.exe --input-file=benchmarks/phylo/set2/tst01.fasta
  --neighborhood=tbr --effort=large --initial-solution=greedy
```

A.2.2 Predictor

In order to predict the best score of some instance we can use the `parsimony_predictor` program with the following options:

- `--deduce=percentage` which tries to deduce relationships with multiple linear regression from the problems of the database and takes into account predictions less than the given percentage knowing the best score. Use for example 1 for 1% accuracy,
- `--predict=percentage` which tries to predict the parsimony score of the instances of the database without knowing the optimal score of the instance to predict. It takes into account predictions less than the given percentage without knowing the best score. Use for example 1 for 1% accuracy,
- `--export` to export the database as a text file
- `--input-file=file` to try to predict the best score of some problem not in the database

```
./parsimony_predictor.exe --predict=1
...
============================================================
SUMMARY
============================================================
problems tested=12270
predicted correctly=10627
percentage= 86.61%
predicted uncorreclty=1635
unsolved=8
total errors=1060630.00
average error=     99.81
minimum error=      0.00
frequency of minimum error=1
maximum error=   4279.31
```

A.2.3 Other software

We also have developed Maximum Parsimony solvers based on:

- a Descent
- an Iterated Local Search (ILS)
- a Genetic Algorithm called *GAMPARS*

- a Memetic Algorithm called *MAMPARS*[2]

Finally the following programs are also available:

- `fitch_score` which given an instance and a newick tree computes the parsimony score of the tree
- `tree_comparator` enables to compare two trees using Path-Relinking

A.3 File formats

We show here the different phylogenetic file formats for some instances.

A.3.1 Sequence group .sg

This file format is very simple and contains one taxon per line. The line starts with the name of the taxon followed by one or more spaces and the residues either in IUPAC[3] nomenclature or using numbers where 0=A, 1=C, ...

```
austroba  0000?10010010000000000000010000000000000000?0000????????00
annonace  00010000000110001000000000100000000000000000000000010000000
myristic  0000?10000010000100000?00100000?00111000000100000001000010
...
convalla  11101000??10000010010000010000000001001000001000100000010000
```

A.3.2 Fasta .fasta

The fasta format identifies each taxon by a line that starts with the symbol > followed by the name of the taxon, a new line and the residues.

```
>austroba
aaaa?caacaacaaaaaaaaaaaaaacaaaaaaaaaaaaaaaaa?aaaa????????aa
>annonace
aaacaaaaaaaccaaacaaaaaaaaacaaaaaaaaaaaaaaaaaaaaaaacaaaaaaa
...
>convalla
cccacaaa??caaaaacaacaaaaacaaaaaaaaacaacaaaaacaaacaaaaacaaaa
```

A.3.3 Phylip .phy

The Phylip format of Joe Felsenstein starts with a line made of two numbers: the number of taxa and the length of a taxon. It is followed by the taxa defined by their names and the residues. This file format is not suitable for taxa that have a long name because the residues must be found on column 11 of the line. The first 10 characters of each line contain the name of the taxon.

```
49 59
austroba  AAAA?CAAAC?CAAAA??ACAA?AAACAAAAAAAACCCAAAA?CAAAAAAAACCAAAAA
annonace  AAAA??AACAAC?AAAAAAAAAAAAAC?C?AA?AAAAAAAAAAA?AAAA????????AA
...
convalla  AAAA?CAACAACAAAAAAAAAAAAAACC??AA?AACCAAAAAAA?AAAA????????AA
```

A.3.4 Nexus .nexus, .nx

The Nexus format (Maddison, Swofford, and Maddison, 1997) is very complex and can contain taxa and eventually trees. In this example we use an identifier for each taxon (`a00000` to `a00048`) and we can establish a correspondence between the identifiers and the names of the taxa using the `TaxLabels` keyword.

```
#NEXUS
BEGIN DATA;
DIMENSIONS  NTAX=49 NCHAR=59 ;
FORMAT DATATYPE = DNA MISSING = ? GAP = - ;
BEGIN TAXA
```

2. based on the work of Adrien Goëffon
3. International Union of Pure and Applied Chemistry

```
TaxLabels chlorant amborell myristic hamameli canellac nymphaea tetracen
caryophy sargento berberid winterac dioscore rafflesi himantan ceratoph gomorteg
trichopo aristolo chenopod >convalla monimiac trochode piperace austroba
hydnorac papavera cabombac dillenia saururac paeoniac sterculi magnolia degeneri
lactorid illiciac eupomati phytolac ranuncul schizand lardizab nelumbon euptelea
fumariac calycant annonace lauracea polygona menisper trimenia ;
END;
MATRIX
a00000     AAAA?CAAAC?CAAAA??ACAA?AAACAAAAAAAACCCAAAA?CAAAAAAAACCAAAAA
a00001     AAAA??AACAAC?AAAAAAAAAAAAAC?C?AA?AAAAAAAAAAA?AAAA????????AA
...
a00048     AAAA?CAACAACAAAAAAAAAAAAAACC??AA?AACCAAAAAAA?AAAA????????AA

;
END ;
```

Index

Bibliography

E Aarts and J. K. Lenstra. *Local search in Combinatorial Optimization*. Princeton University Press, 2003. 27

E. H. L. Aarts and P. J. M. Van Laarhoven. Statistical cooling: A general approach to combinatorial optimization problems. *Philips Journal of Research*, 40:193–226, 1985. 42

D. Abramson, M. Krishnamoorthy, and H. Dang. Simulated annealing cooling schedules for the school timetabling problem. *Asia-Pacific Journal of Operational Research*, 16(1):1–22, 1999. 42

B. J. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001. 28

A. Andreatta and C. C. Ribeiro. Heuristics for the phylogeny problem. *Journal of Heuristics*, 8:429–447, 2002. 26, 28, 41, 73

D. A. Bader, V. P. Chandu, and M. Yan. ExactMP: An efficient parallel exact solver for phylogenetic tree reconstruction using maximum parsimony. In *Parallel Processing, 2006. ICPP 2006. International Conference on*, pages 65–73. IEEE, 2006. 35, 73

D. Barker. LVB: Parsimony and simulated annealing in the search for phylogenetic trees. *Bioinformatics Applications Note*, 20 (2):274–275, August 2003. 31, 41, 43, 48, 50, 73

D. Barker. LVB homepage, 2012. URL `http://biology.st-andrews.ac.uk/cegg/lvb.aspx`. 41, 43, 46, 48, 50

A. Ben-Dor, B. Chor, D. Graur, R. Ophir, and D. Pelleg. Constructing phylogenies from quartets: Elucidation of eutherian superordinal relationships. *Journal of Computational Biology*, 5(3):377–390, 1998. 36

J. Blazewicz, P. Formanowicz, P. Kedziora, P. Marciniak, and P. Taront. Adaptive memory programming: Local search parallel algorithms for phylogenetic tree construction. *Annals of Operations Research*, 183(1):75–94, 2011. 35

G. E. Blelloch and B. M. Maggs. Parallel algorithms. *ACM Computing Surveys (CSUR)*, 28(1):51–54, 1996. 35

R. W. Burkhardt and R. Wellington. *The Spirit of System: Lamarck and Evolutionary Biology: Now with" Lamarck in 1995"*. Belknap Press, 1995. 15

J.H. Camin and R.R Sokal. A method for deducing branching sequences in phylogeny. *Evolution*, pages 311–326, 1965. 16

W. Cancino and A. Delbem. A multi-objective evolutionary approach for phylogenetic inference. In *Evolutionary Multi-Criterion Optimization*, pages 428–442. Springer, 2007. 36

V. Cerny. A thermodynamic approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, 1985. 31, 39

I. Charon and O. Hudry. The noising methods: a survey. In *Essays and Surveys in Metaheuristics*, pages 245–261. Kluwer, 2002. 31

M. W. Chase, D. E. Soltis, R. G. Olmstead, D. Morgan, D. H. Les, B. D. Mishler, M. R. Duvall, R. A. Price, H. G. Hills, Y. Qiu, K. A. Kron, J. H. Rettig, E. Conti, J. D. Palmer, J. R. Manhart, K. J. Sytsma, and al. Phylogenetics of seed plants: An analysis of nucleotide sequences from the plastid gene rbcl. *Annals of the Missouri Botanical Garden*, 80(3):528–580, 1993. 54

D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The combinatorial design approach to automatic test generation. *IEEE Software*, 13:83–88, 1996. 45, 50

C. J. Colbourn. Combinatorial aspects of covering arrays. *Le Matematiche*, 58:121–167, 2004. 45

Ch. Darwin. *On the Origin of Species*. John Murray (London), 1859. 15

Ch. Darwin. *The Origin of Species by Means of Natural Selection: Or, The Preservation of Favoured Races in the Struggle for Life and the Descent of Man and Selection in Relation to Sex*. Modern Library, 1872. 15

J. De Laet. Pseudocode for some tree search algorithms in phylogenetics. Technical report, Koninklijk Belgisch Instituut voor Natuurwetenschappe, 2005. 48

W. A. de Landgraaf, A. E. Eiben, and V. Nannen. Parameter calibration using meta-algorithms. In *In proceedings of the IEEE Congress on Evolutionary Computation*, pages 71–78. IEEE Press, 2007. 45

Z. Du, F. Lin, and U. W. Roshan. Reconstruction of large phylogenetic trees: a parallel approach. *Computational Biology and Chemistry*, 29(4):273–280, 2005. 35

A. W. F. Edwards and C. L. L. Sforza. The reconstruction of evolution. *Heredity*, 18, 1963. 16

G. Estrin. Reconfigurable computer origins: the ucla fixed-plus-variable (f+ v) structure computer. *Annals of the History of Computing, IEEE*, 24(4):3–9, 2002. 36

J. S. Farris. Phylogenetic analysis under Dollo's Law. *Systematic Zoology*, 26:77–88, 1977. 16

J. Felsenstein. Maximum Likelihood and Minimum-Steps Methods for Estimating Evolutionary Trees from Data on Discrete Characters. *Systematic Zoology*, 22(3):240–249, 1973. doi: 10.2307/2412304. 18

J. Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of Molecular Evolution*, 17(6): 368–376, 1981. 16

J. Felsenstein. Inferring phylogenies. *Sunderland, Massachusetts: Sinauer Associates*, 2004. 16, 19

W. Fitch. Toward defining the course of evolution : Minimum change for a specific tree topology. *Systematic Zoology*, 20(4): 406–416, 1971. 16, 18, 20, 22

W. Fitch and E. Margoliash. Construction of phylogenetic trees. *Sience*, 155(3760):279–284, 1967. 18

P. G. Foster and D. A. Hickey. Compositional bias may affect both dna-based and protein-based phylogenetic reconstructions. *Journal of Molecular Evolution*, 48(3):284–290, 1999. 19

L. R. Foulds and R. L. Graham. The Steiner tree problem in phylogeny is NP-complete. *Advances in Applied Mathematics*, 3: 43–49, 1982. 22, 24

G. Ganapathy, V. Ramachandran, and T. Warnow. On contract-and-refine transformations between phylogenetic trees. In *Proceedings of 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 900–909, 2004. ISBN 0-89871-558-X. 27, 29

M. Garey and D. Johnson. The rectilinear steiner tree problem is np-complete. *SIAM Journal on Applied Mathematics*, 32(4): 826–834, 1977. 12, 23

O. Gascuel. On the optimization principle in phylogenetic analysis and the minimum-evolution criterion. *Molecular Biology and Evolution*, 17(3):401, 2000. 40

G. Giribet. Efficient Tree Searches with Available Algorithms. *Evolutionary Bioinformatics*, 3:341–356, 2007. URL `http://la-press.com/article.php?article_id=431`. 25

D. S. Gladstein. Efficient incremental character optimization. *Cladistics*, 13(1-2):21–26, 1997. 49, 73, 74

F. Glover. Tabu searchâpart i. *ORSA Journal on Computing*, 1(3):190–206, 1989. 32, 35

F. Glover. A template for scatter search and path relinking. In *Artificial evolution*, pages 1–51. Springer, 1998. 51

F. Glover and M. Laguna. *Tabu Search*, volume 3 of *Handbook of Combinatorial Optimization*. Kluwer Academic Publishers, d-z edition, July-August 1990. 32, 35

F. Glover and C. McMillan. *The General Employee Scheduling Problem. An Integration of MS and AI*, volume 13 of *Handbook of Combinatorial Optimization*. Elsevier, 1986. 32

F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000. 51

A. Goëffon. *Nouvelles Heuristiques de Voisinage et Mémétiques pour le Probléme Maximum de Parcimonie*. PhD thesis, Université d'Angers, november 2006. 23, 24, 43, 45, 48, 49, 50, 82

A. Goëffon, J.-M. Richer, and J. K. Hao. Progressive tree neighborhood applied to the maximum parsimony problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 5(1):136–145, 2008. 30, 35, 40, 82

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, volume 18 of *Artificial Intelligence*. Addison-Wesley, 1989. ISBN 978-0201157673. 33

P. A. Goloboff. Nona, version 1.6, 1997. 26, 37

P. A. Goloboff. Techniques for analyzing large data sets. In R. DeSalle R, G. Giribet, and W. Wheeler eds., editors, *Techniques in Molecular Systematics and Evolution*, pages 70–79. Brikhäuser Verlag, Basel, 2002. 32, 37, 48

P. A. Goloboff, J. S. Farris, and K. C. Nixon. Tnt, a free program for phylogenetic analysis. *Cladistics*, 24(5):774–786, 2008a. 32, 37, 43, 48, 61

P.A. Goloboff. Character optimization and calculation of tree lengths. *Cladistics*, 9:433–436, 1993. 22, 73, 74

P.A. Goloboff. Oblong, a program to analyse phylogenomic data sets with millions of characters, requiring negligible amounts of ram. *Cladistics*, 30(3):273–281, 2014. ISSN 1096-0031. doi: 10.1111/cla.12056. URL http://dx.doi.org/10.1111/cla.12056. 81

P.A. Goloboff, J.M. Carpenter, J.S. Arias, and D.R. Esquivel. Weighting against homoplasy improves phylogenetic analysis of morphological data sets. *Cladistics*, 24(5):758–773, 2008b. 62

S. J. Gould. *The Structure of Evolutionary Theory*. Belknap Press, 2002. 15

A. Gunawan, H. C. Lau, and Lindawati. Fine-tuning algorithm parameters using the design of experiments. *Lecture Notes in Computer Science*, 6683:131–145, 2011. 45

D. M. Gusfield. *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology*. University of Cambridge, 1st. edition, 1997. ISBN 0-521-58519-8. 23

E. Haeckel. *Generelle Morphologie der Organismen*, volume bd. 1. Georg Riemer (Berlin), 1866. 15

P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable neighbourhood search: Methods and applications. *Annals of Operations Research*, 175(1):367–407, 2010. 29

M. D. Hendy and D. D. Penny. Branch and bound algorithms to determine minimal evolutionary trees. *Mathematical Biosciences*, 59(2):277–290, 1982. 26

W. Hennig. *Phylogenetic Systematic*. Phylogeny. University of Illinois, 1966. ISBN 0-252-06814-9. 12, 16

J. H. Holland. *Adaption in Natural and Artificial Systems*. Complex Adaptive Systems. The University of Michigan Press, 1975. ISBN 0-262-58111-6. 33

H. H. Hoos and T. Stützle. *Stochastic Local Search, Foundations and Applications*. Morgan Kaufmann Publishers, 2005. 34

D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part I, graph partitioning. *Operations Research*, 37(6):865–892, 1989. 39

D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon. Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning. *Operations Research*, 39(3):378–406, 1991. 39

S. Kasap and K. Benkrid. High performance phylogenetic analysis with maximum parsimony on reconfigurable hardware. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 19(5):796–808, 2011. 36

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983. 31, 39

A. Kluge and J. Farris. Quantitative phyletics and the evolution of anurans. *Systematic Zoology*, 18(1):1–32, 1969. 12

P. Lagassé, L. Goldman, A. Hobson, and S. R. Norton. *The Columbia Encyclopedia*. Columbia University Press, 2000. 16

P. O. Lewis. A genetic algorithm for maximum likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution*, 15(3):277–283, 1998. 34

Y. Lin, S. Fang, and J. L. Thorne. A tabu search algorithm for maximum parsimony phylogeny inference. *European Journal of Operational Research*, 176(3):1908–1917, 2007. 32

Y. Lin, V. Rajan, and B. M. E. Moret. A metric for phylogenetic trees based on matching. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(4):1014–1022, 2012. 56, 57

H. R. Lourenco, O. Martin, and T. Stützle. Iterated local search. In *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*, pages 321–353. Kluwer Academic Publishers, 2002. 31

Z. Lü, J. K. Hao, and F. Glover. Neighborhood analysis: A case study on curriculum-based course timetabling. *Journal of Heuristics*, 17(2), 2011. 41

M. Luckow and R.A. Pimentel. An empirical comparison of numerical wagner computer programs. *Cladistics*, 1(1):47–66, 1985. ISSN 1096-0031. doi: 10.1111/j.1096-0031.1985.tb00410.x. URL http://dx.doi.org/10.1111/j.1096-0031.1985.tb00410.x. 61

D.R. Maddison, D.L. Swofford, and W.P. Maddison. Nexus: An extensible file format for systematic information. *Systematic Biology*, 46(4):590–621, 1997. doi: 10.1093/sysbio/46.4.590. 86

H. Matsuda. Protein phylogenetic inference using maximum likelihood with a genetic algorithm. *Proceedings of 1st Pacific Symposium on Biocomputing*, 3(1):512–523, 1996. 34

G. Mendel. Experiments in plant hybridization (1865). *Read at the February*, 8:3–47, 1865. 15

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, and A. H. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953. 31, 39

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24(11):1097–1100, 1997. 29

D. A. Morrison. Increasing the efficiency of searches for the maximum likelihood tree in a phylogenetic analysis of up to 150 nucleotide sequences. *Systematic Biology*, 56(6):988–1010, 2007. ISSN 10635157, 1076836X. URL http://www.jstor.org/stable/20143107. 62

P. Moscato. Memetic algorithms: A short introduction. In *New Ideas in Optimization*, pages 219–234. McGraw-Hill, London, 1999. 34

Y. Murakami and S. Jones. Sharp2: Protein–protein interaction predictions using patch analysis. *Bioinformatics*, 22(14):1794–1795, 2006. 12

K. C. Nixon. The parsimony ratchet, a new method for rapid parsimony analysis. *Cladistics*, 15(4):407–414, 1999. 22, 37

P.A. Nuin, Z. Wang, and E.R. Tillier. The accuracy of several multiple sequence alignment programs for proteins. *BMC Bioinformatics*, 7(1):1–18, 2006. 38

N. R. Pace. A molecular view of microbial diversity and the biosphere. *Science*, 276(5313):734–740, 1997. 12

A. S. Packard. Lamarck, the founder of evolution. his life and work, with translations of his writings on organic evolution. london, new york & bombay, 1901. 15

Ch. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Dover publications, 1998. 27

J. Pearsall and P. Hanks. *The New Oxford Dictionary of English*. Clarendon Press, 1998. 16

E. M. Phizicky and S. Fields. Protein-protein interactions: Methods for detection and analysis. *Microbiological Reviews*, 59(1): 94–123, 1995. 12

S. Pirkwieser and G.R. Raidl. Finding consensus trees by evolutionary, variable neighborhood search, and hybrid algorithms. In *Proceedings of Generic and Evolutionary Computation Conference (GECCO'08)*, 2008. 30

M. G. C. Resende and C. C. Ribeiro. Grasp: Greedy randomized adaptive search procedures. In Edmund K. Burke and Graham Kendall, editors, *Search Methodologies*, pages 287–312. Springer US, 2014. 51

C. C. Ribeiro and D. S. Vianna. A genetic algorithm for the phylogeny problem using an optimized crossover strategy based on path-relinking. In *Proc. of 2nd Bresil Workshop on Bioinformatics*, pages 97–102, 2003. 45

C. C. Ribeiro and D. S. Vianna. A GRASP/VND heuristic for the phylogeny problem using a new neighborhood structure. *International Transactions in Operational Research*, 12(3):325–338, 2005. 29, 33, 35, 41, 45, 61, 73

C. C. Ribeiro and D. S. Vianna. A hybrid genetic algorithm for the phylogeny problem using path-relinking as a progressive crossover strategy. *International Transactions in Operational Research*, 16(5):641–657, 2009. 41, 43, 45, 48, 49, 50, 51, 52, 55, 56, 73

J-M. Richer. Three new techniques to improve phylogenetic reconstruction with maximum parsimony. Technical report, LERIA, University of Angers, France, 2008. 74

J-M. Richer. Improvevement of fitch function for maximum parsimony in phylogenetic reconstruction with intel avx2 assembler instructions. Technical report, LERIA, University of Angers, France, 2013. URL http://www.info.univ-angers.fr/pub/richer/pub/tr20130624-1.pdf. 32, 75

J.-M. Richer, A. Goëffon, and J. K. Hao. A memetic algorithm for phylogenetic reconstruction with maximum parsimony. In *EvoBIO*, volume 5483 of *Lecture Notes in Computer Science*, pages 164–175. Springer, 2009. 32, 34, 73

D. Robilliard, V. Marion-Poty, and Cyril F. Population parallel GP on the g80 GPU. In *Genetic Programming*, pages 98–109. 2008. URL http://dx.doi.org/10.1007/978-3-540-78671-9_9. 79

D. Robinson. Comparison of labeled trees with valency three. *Journal of Combinatorial Theory, Series B*, 11:105–119, 1971. 28

D. F. Robinson and L. R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981. 56

E. Rodriguez-Tello and J. Torres-Jimenez. Memetic algorithms for constructing binary covering arrays of strength three. *Lecture Notes in Computer Science*, 5975:86–97, 2010. 46

E. Rodriguez-Tello, J. K. Hao, and J. Torres-Jimenez. An effective two-stage simulated annealing algorithm for the minimum linear arrangement problem. *Computers & Operations Research*, 35(10):3331–3346, 2008. 42

F. Ronquist. Fast fitch-parsimony algorithms for large data sets. *Cladistics*, 14(4):387–400, 1998. 75

F. Ronquist. Fast fitch-parsimony algorithms for large data sets. *Cladistics*, 14:387–400, 2000. 73

A. Rzhetsky and M. Nei. Theoretical foundation of the minimum-evolution method of phylogenetic inference. *Molecular Biology and Evolution*, 10(5):1073–1095, 1993. 18

N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 4(4):406–425, 1987. 17, 40

J. Sanders and E. Kandrot. *CUDA by Example: An Introduction to General-Purpose GPU Programming*. Addison-Wesley Professional, 2010. 76

D. Sankoff and P. Rousseau. Locating the vertices of a Steiner tree in an arbitrary metric space. *Mathematical Programming*, 2: 240–246, 1975. 24

K. Seridi, L. Jourdan, and E. G. Talbi. Multi-objective path relinking for biclustering: application to microarray data. In *EMO'2013 Evolutionary Multi-objective Optimization*, pages 200–214, 2013. 51

A. Shaban-Nejad and V. Haarslev. Ontology-inferred phylogeny reconstruction for analyzing the evolutionary relationships between species: Ontological inference versus cladistics. In *BioInformatics and BioEngineering, 2008. BIBE 2008. 8th IEEE International Conference on*, pages 1–7. IEEE, 2008. 16

P.H.A. Sneath and R.R. Sokal. *Numerical Taxonomy. The Principles and Practice of Numerical Classification*. Freeman, 1973. 40

Q. Snell, M. Whiting, M. Clement, and D. McLaughlin. Parallel phylogenetic inference. In *SC2000: High Performance Networking and Computing. Dallas Convention Center, Dallas, TX, USA*, pages 62–62. IEEE ACM, 2000. 35

R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Science Bulletin*, 38:1409–1438, 1958. 17

D. L. Swofford and G. J. Olsen. Phylogeny reconstruction. In *Molecular Systematics*, pages 411–501. Sinauer Associates, Inc., Sunderland, USA, 1990. 28, 31

D. L. Swofford, G. J. Olsen, P-J. Waddell, and D. M. Hillis. Phylogenetic inference. In *Molecular Systematics (2nd edition)*, pages 407–514. Sunderland, USA, 1996a. 25

D.L. Swofford. Paup*: Phylogenetic analysis using parsimony 4.0 beta, 2002. *Molecular Systematics*, pages 411–501, 1990. 16

D.L. Swofford, G.J. Olsen, P.J. Waddell, and D.M. Hillis. Chapter 11: Phylogenetic inference. In *Molecular Systematics*, volume 2. Sinauer Associates, Inc., Sunderland, 1996b. 26

J.D. Thompson, T. Gibson, and D.G. Higgins. Multiple Sequence Alignment Using ClustalW and ClustalX. *Current protocols in bioinformatics / editoral board, Andreas D. Baxevanis ... [et al.]*, Chapter 2, August 2002. 16

N. L. J. Ulder, E. H. L. Aarts, H. Bandelt, P. J. M. van Laarhoven, and E. Pesch. Genetic local search algorithms for the traveling salesman problem. In *Parallel Problem Solving from Nature - PPSN I*, volume 496 of *Lecture Notes in Computer Science*, pages 109–116. Springer, 1991. 34

P. J. M. Van Laarhoven and E. H. L. Aarts. *Simulated annealing: Theory and applications*. Kluwer Academic Publishers, 1988. 42

K. E. Vázquez-Ortiz and E. Rodríguez-Tello. Metaheuristics for the maximum parsimony problem. *Computacional Intelligence and Bioinformatics*, pages 105–113, 2011. 31, 32, 43

G. V. R. Viana, F. A. C. Gomes, C. E. Ferreira, and C. N. Meneses. Parallelisation of a multi-neighbourhood local search heuristic for a phylogeny problem. *International Journal of Bioinformatics Research and Applications*, 5(2):163–177, 2009. 35

W. H. Wagner. Problems in the classification of ferns. *in Recent advances in botany*, pages 841–844, 1961. 16, 26

Y. Wang, Z. Lü, F. Glover, and J. K. Hao. Path relinking for unconstrained binary quadratic programming. *European Journal of Operational Research*, 223(3):595–604, 2012. 51

M. S. Waterman and T. F. Smith. On the similarity of dendrograms. *Journal of Theoretical Biology*, 73(4):784–900, 1978. 28

G. Weber, L. Ohno-Machado, and S. Shieber. Representation in stochastic search for phylogenetic tree reconstruction. *Biomedical Informatics*, 39:43–50, 2005. 26, 27

W.C. Wheeler, N. Lucaroni, L. Hong, L.M. Crowley, and A. Varón. POY version 5: phylogenetic analysis using dynamic homologies under multiple optimality criteria. *Cladistics*, 31(2):189–196, 2015. 37

W. T. J. White and B. R. Holland. Faster exact maximum parsimony search with xmp. *Bioinformatics*, 27(10):1359–1367, 2011. 35, 62

B. Y. Wu, K. Chao, and C. Y. Tang. Approximation and exact algorithms for constructing minimum ultrametric trees from distance matrices. *Combinatorial Optimization*, 3:199–212, 1999. 26

Y. Xin and G. S. Xiao. *Linear Regression Analysis: Theory and Computing*. World Scientific, 2009. 60

J. Xiong. *Essential Bioinformatics*. Cambridge University Press, 1st. edition, 2006. ISBN 978-0521600828. 13, 25

M. Yan and D. A. Bader. Fast character optimization in parsimony phylogeny reconstruction. Technical Report TR-CS-2003-53, University of New Mexico, Albuquerque, NM, USA, 2003. 73, 74

Y. Zhou, J. Xie, and H. Zheng. A hybrid bat algorithm with path relinking for capacitated vehicle routing problem, 2013. 51

Thèse de Doctorat

Karla E. VAZQUEZ ORTIZ

Méthodes avancées pour la résolution du problème de maximum parcimonie

Advanced methods to solve the maximum parsimony problem

Résumé
La reconstruction phylogénétique est considérée comme un élément central de divers domaines comme l'écologie, la biologie et la physiologie moléculaire pour lesquels les relations généalogiques entre séquences d'espèces ou de gènes, représentées sous forme d'arbres, peuvent apporter des éclairages significatifs à la compréhension de phénomènes biologiques. Le problème de Maximum de Parcimonie est une approche importante pour résoudre la reconstruction phylogénétique en se basant sur un critère d'optimalité pour lequel l'arbre comprenant le moins de mutations est préféré.
Dans cette thèse nous proposons différentes méthodes pour s'attaquer à la nature combinatoire de ce problème NP-complet. Premièrement, nous présentons un algorithme de Recuit Simulé compétitif qui nous a permis de trouver des solutions de meilleure qualité pour un ensemble de problèmes. Deuxièmement, nous proposons une nouvelle technique de Path-Relinking qui semble intéressante pour comparer des arbres mais pas pour trouver des solutions de meilleure qualité. Troisièmement, nous donnons le code d'une implantation sur GPU de la fonction objectif dont l'intérêt est de réduire le temps d'exécution de la recherche pour des instances dont la longueur des séquences est importante. Finalement, nous introduisons un prédicteur capable d'estimer le score optimum pour un vaste ensemble d'instances avec une très grande précision.

Mots clés
Reconstruction Phylogénétique, Maximum Parcimonie, Optimisation Combinatoire, Recuit Simulé

Abstract
Phylogenetic reconstruction is considered a central underpinning of diverse fields like ecology, molecular biology and physiology where genealogical relationships of species or gene sequences represented as trees can provide the most meaningful insights into biology. Maximum Parsimony (MP) is an important approach to solve the phylogenetic reconstruction based on an optimality criterion under which the tree that minimizes the total number of genetic transformations is preferred.
In this thesis we propose different methods to cope with the combinatorial nature of this NP-complete problem. First we present a competitive Simulated Annealing algorithm which helped us find trees of better parsimony score than the ones that were known for a set of instances. Second, we propose a Path-Relinking technique that appears to be suitable for tree comparison but not for finding trees of better quality. Third, we give a GPU implementation of the objective function of the problem that can reduce the runtime for instances that have an important number of residues per taxon.
Finally, we introduce a predictor that is able to estimate the best parsimony score of a huge set of instances with a high accuracy.

Key Words
Phylogenetics, Maximum Parsimony, Combinatorial Optimization, Simulated Annealing.