CHAPTER 1

HEURISTIC METHODS FOR PHYLOGENETIC RECONSTRUCTION WITH MAXIMUM PARSIMONY

In this chapter we explain how metaheuristics like local search, genetic and memetic algorithms are used for phylogenetic reconstruction using Maximum Parsimony. We review some of the main concepts used to improve the search of a good solution which are inherited from the Operational Research and Combinatorial Optimization communities.

—Adrien Goëffon, Jean-Michel Richer and Jin-Kao Hao

1.1 INTRODUCTION

Maximum Parsimony (MP) is a character-based approach that relies on the work of the german entomologist Willy Hennig (1913-1976). Although Hennig's work has generated significant controversy, the principles that underlie what was later called cladistics, laid the basis for a convenient and powerful method for the analysis of molecular data with the use of computers. For more details about the history of MP see [9] (p. 136). Cladistics also referred as phylogenetic systematics can be viewed as a philosophy of classification that arranges organisms only by their order of branching in an evolutionary tree. The leaves of the tree are labelled with the OTUs (Operational Taxonomic Unit) of the problem. Ideally, the trees (or cladograms) that

result from a MP analysis show the evolution of synapomorphies (derived character states inherited from the most common ancestor) between species. Many different cladograms can exist for a given set of taxa but the MP criterion imposes to chose the ones with the fewest changes.

1.2 DEFINITIONS, FORMAL BACKGROUND

1.2.1 Parsimony and Maximum Parsimony

With parsimony methods, each position (or site) in the multiple alignment is considered separately. First note that there are different parsimony optimality criterion 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. In the rest of this chapter we are only interested in Fitch (or unweighted) parsimony for which all mutations are given the same weight of one unit.

The input of the problem consists of a set L of n sequences of the same length m expressed over an alphabet Σ, where $\Sigma = \{-, A, C, G, T, ?\}$[1] is composed of four nucleotides, the gap symbol - (if L is the result of a multiple alignment) and eventually the missing character symbol: ?. The second input of the problem is a binary rooted or unrooted tree whose leaves are labelled with the sequences of L. Other nodes of the tree, called internal nodes, have two descendants (see fig. 1.1). We can then define two problems called *small* and *large* parsimony problems.

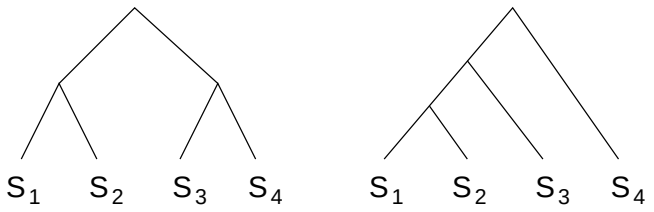


Figure 1.1: Two topologies for a binary rooted tree of 4 sequences

Definition 1 (Small parsimony problem) *Given a set L of n sequences of length m and a tree T whose leaves are labelled with sequences of L, find the score of parsimony of T*

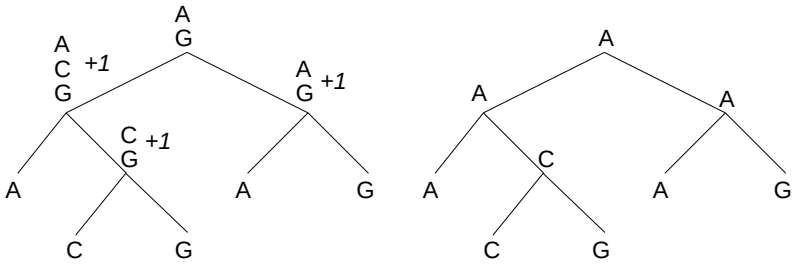
In order to compute the overall cost (or score) of a tree (also known as *tree length*), Fitch's algorithm [11] gradually moves back from the leaves to the root and computes hypothetical ancestral taxa. This is often referred to as the first-pass of the algorithm (see algorithm 1.4 for a more formal description). At each position of an internal node

[1] It is also possible to use protein sequences with a 20 letters alphabet of amino-acids.

a set of bases is assigned. If two descendants x and y of an internal node v have some bases in common they are assigned to the internal node $L_v = L_x \cap L_y$. Otherwise all bases of both descendants are assigned to the parent $L_v = L_x \cup L_y$ and a cost of one unit is added to the overall score of the tree (see fig. 1.2). 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.

Definition 2 (Large parsimony problem or Maximum Parsimony problem) *Given a set L of n sequences of length m, find a most parsimonious tree T, i.e. a tree with minimum score.*

The cost of a tree can be computed in polynomial time [11]. A rooted binary tree of n leaves has $n-1$ internal nodes, thus the complexity of the small parsimony problem is $O(n \times m)$. 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 [12]. 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 [21, 36].



a) first−pass from leaves to root

b) second−pass from root to leaves

Figure 1.2: First pass and second pass for a tree of score of 3 under Fitch's optimality criterion

1.3 METHODS

1.3.1 Combinatorial Optimization

Combinatorial Optimization problems consist in finding the *best* object also called the optimum or optimal solution in a finite (or possibly countably infinite) set of objects called solutions. This class of problem is generally NP-complete [15] which

roughly means that the computing time needed by an algorithm to find an optimal solution would increase exponentially with the size of the problem to solve. Due to the importance of these problems, many algorithms have been developed. These algorithms are of twofold:

- *exact* algorithms are guaranteed to find the best solution but might need exponential computation time, they try to optimize the search by ignoring configurations that can be identified as inappropriate during the search.
- *approximate* methods trade optimality for efficiency by examining an appropriate subset of the solutions in order to find a good near-optimal solution.

1.3.2 Exact approach

1.3.2.1 exhaustive enumeration The simplest algorithm that can be designed to find the most parsimonious tree(s) is to generate all possible trees and compute their parsimony score. However tree searches are extremely difficult because the number of possible trees grows exponentially with the number of taxa (see table 1.1).

Table 1.1: number of unrooted and rooted binary trees

number of taxa	number of unrooted trees	number of rooted trees
10	2.0e+06	3.4e+07
20	2.2e+20	8.2e+21
30	8.6e+36	4.9e+38
40	1.3e+55	1.0e+57
50	2.8e+74	2.7e+76
80	2.1e+137	3.4e+139
n	$\prod_{i=3}^{n}(2i-5)$	$\prod_{i=2}^{n}(2i-3)$

1.3.2.2 branch and bound A first alternative to tackle the complexity of MP is the branch and bound (B&B) algorithm [23]. We first generate a tree, not necessarily optimal, and compute its parsimony score which will serve as an upper bound. We then start the construction of a new tree, initially empty, and add a new taxon at each iteration. Each new taxon is put on all possible branches of the previous trees and generates a set of new trees which are put in a list. The trees which have a parsimony score greater than the upper bound are withdrawn from the list. The main drawback of the B&B algorithm is that the list of trees is too important in order for the algorithm to be efficient, that is why the algorithm can only be applied on a set of less than 20 taxa.

1.3.3 Local Search methods

Due to the amount of trees to evaluate and the inefficiency of exact methods, it is preferable to use approximate approaches for inferring large phylogenetic trees.

Local Search (LS) uses iterative improvements to seek for solutions of better quality [25]. A LS algorithm consists of four essentials parts:

- a search space S composed of a set of candidate solutions
- an evaluation function $f(s)$ of a solution $s \in S$, also called *fitness* function, in order to assess the quality of a solution
- a neighborhood function $N(s) \subset S$ in order to define for each solution a subset of solutions which can be obtained by slightly modifying the current solution
- a transition strategy to accept or reject a neighboring solution

Typically, a LS algorithm (see algorithm 1.1) starts from an initial solution s and then iteratively replaces the current solution by a neighbor $s' \in N(s)$ of better quality, until no improving neighbor can be found. This process is sometimes called a *replication* in the MP literature and a *descent* for the optimization community. The replacement of the current solution favors neighbors of better quality with the intent to progressively improve the quality of the solution.

Algorithm 1.1

descent (S, f, N)
  s := chosse or generate an initial solution $\in S$
  for a given number of iterations i ***do***
    find $s' \in N(s)$ such that $f(s') < f(s)$
    $s := s'$
  end for
  return s

In the case of MP, the search space is the set of all possible binary (rooted or unrooted) trees and the fitness function is the parsimony score of a tree. Finding the most parsimonious tree is then a *minimization* problem. A solution s_1 is better than s_2 if $f(s_1) < f(s_2)$ and the solution s_2 is said to be of lower quality than s_1.

1.3.3.1 generation of the initial solution

We can generate an initial solution by two means. We can either generate a solution by randomly selecting taxa that are put on any branch of the generated tree or use an approximate method called *stepwise addition*. The stepwise addition, also known as Wagner trees, is close to the B&B algorithm but only keeps track of one most parsimonious tree. Each new taxon is inserted on all possible branches but only the tree which gives the best score is retained. The order in which the taxa are successively added plays an important role and the method will generally produce suboptimal trees. Trees obtained by B&B generally provide final solutions of better quality than trees generated through a random process.

1.3.3.2 the local optimum problem

The main drawback of LS is that it can often get stuck in a *local optimum*: namely, a solution which is not the best but that is locally better than its neighbors. More formally, a local optimum $s^\diamond$ is such that $\forall s' \in N(s^\diamond), f(s^\diamond) \leq f(s')$ and $f(s^\diamond) > f(s^\star)$.

In order to escape from a local optimum several techniques have been designed. Some techniques allow the selection of neighbors of same or lower quality than the current solution, while others modify the evaluation function or pertub the current solution. For example:

1. the *side-walk descent* allows to chose improving or equivalent neighbors during a certain number of iterations contrary to a pure descent algorithm which only accepts strictly improving neighbors; this less restrictive condition allows to escape from a local optimum and provides more randomness to the process,
2. the *random walk* is a similar process which offers the possibility to accept deteriorating neighbors (i.e. of lower quality) with a given probability,
3. the well-known *simulated annealing* method (see section 1.3.3.4) is a specific random walk with a non-constant probability to accept neighbors of lower quality, depending on the importance of the deterioration as well as the progress of the search.

The noising techniques can either modify the current solution or modify the fitness function for a given number of iterations. The modification of the current solution is known as *Iterated Local Search* (ILS) [29], while the modification of the fitness function applied to MP is known as *Parsimony Ratchet* [36, 26]. When a local optimum $s^\diamond$ is reached, the Ratchet noises the evaluation function: the weights of a proportion of the characters (10-15%) can be increased or some characters can be eliminated. A second descent is performed from $s^\diamond$ using the noising evaluation function f'. Actually, $s^\diamond$ is generally not a local optimum in (S, f'), so the configuration is improved in this new search space (but deteriorated considering the initial fitness function). The solution s' obtained from a descent using f' is the starting point for a new LS process, using the initial score function f. This process is repeated during a fixed number of iterations.

1.3.3.3 Neighborhoods

In the case of the MP problem different neighborhoods have been conceived which are identified under the term *branch-swapping*. They greatly influence the search for the best solution and are briefly recalled here.

Traditional neighborhoods. Three complementary neighborhoods are traditionally used in phylogenetic reconstruction: NNI, SPR, and TBR (see figure 1.3). Depending on the context and the community, these acronyms denominate either the local search algorithm using the related neighborhood or only the neighborhood function.

- **NNI** (*Nearest Neighbor Interchange*) [46] consists in swapping two subtrees which are separated by a branch. This is a small neighborhood since each tree of n leaves has $(2n - 6)$ NNI neighbors [42] ($n - 3$ internal branches and two possible swaps for each branch). An extension to NNI has been proposed by [13, 14] into a parametric neighborhood p-ECR which shuffles p adjacent branches. In particular, 1-ECR is equivalent to NNI.

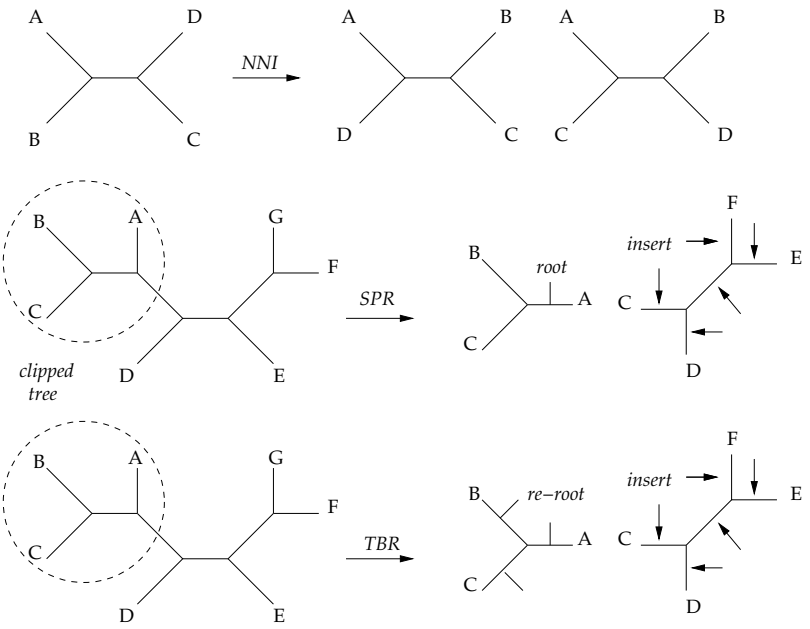


Figure 1.3: Traditional neighborhoods: NNI, SPR, TBR

- A **SPR** move (*Subtree Pruning Regrafting*) [45] cuts a branch and leaves two trees: the clipped tree and the residual tree. The pruned tree can then be re-grafted on each branch of the residual tree to obtain a new topology. We can generate $2(n-3)(2n-7)$ SPR rearrangements [1]. A particular case of SPR which moves only one leave in the tree is called STEP.

- **TBR** (*Tree Bisection Reconnection*) [45] is a larger neighborhood which breaks the tree into two subtrees and reconnects the re-rooted clipped tree to any branch of the residual tree. The number of TBR neighbors depends on the tree topology, but it is at least equal to $(2n-3)(n-3)^2$.

An important property is that these three neighborhoods are imbricated: $NNI \subseteq SPR \subseteq TBR$, and have three distinct levels of complexity, respectively: $O(n), O(n^2)$ and $O(n^3)$. At least one of these neighborhoods are used in every phylogenetic reconstruction software based on branch-swapping.

The complexity of the LS algorithm thus mainly depends on the complexity of the neighborhood. For example, LS+SPR has a worst complexity of $O(i \times n^3 \times m)$[2] where i is the number of iterations of the LS algorithm.

[2]the complexity of the construction of a tree is $O(n \times m)$ and the complexity of SPR is $O(n^2)$.

Variable neighborhoods. A neighborhood affects two main factors of a LS algorithm: the quality of the solutions found and the computation time. Small complexity neighborhoods like NNI enable to perform a quick search and are time scalable with the size of instances. When the number of taxa is increased, the size of the search space only grows linearly. The main drawback is that the current solution does not undergo enough modifications and there is a high probability to observe a premature convergence to a local optimum. For a large size neighborhood like TBR, the number of neighbors to evaluate makes the search more computationally intensive but the improvements can be important. It follows that SPR, as a medium size neighborhood, is often used by descent algorithms as it permits to obtain solutions of better quality than NNI and with less computation time than TBR.

Based on the observation that the size of a neighborhood influences the search, [32] have introduced the notion of *Variable Neighborhood Search* (VNS). The principle of the VNS metaheuristics is to successively use different neighborhoods during a descent by starting from a small size neighborhood and once the search is stuck in a local optimum, one uses a neighborhood of larger size in order to allow important modifications of the current solution.

For example, an application of VNS to MP was proprosed by Ribeiro and Vianna [40] with the use of two neighborhoods: SPR and 2-SPR (where l-SPR is the composition of l SPR transformations). The algorithm starts with a SPR descent and switches to 2-SPR when a local optimum is found. It obtains good results despite an important computation time. In practice, l does not exceed 2 since the l-SPR neighborhood, with a size of $O(n^{2^l})$ is rapidly too large.

Progressive neighborhood. Based on the observation that only important topological modifications of the tree are performed at the beginning of the descent, we have proposed a *Progressive Neighborhood* (PN) [19], 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 non pertinent configurations. Recently PN has been used by [37] to find consensus trees of high quality.

In order to make the neighborhood evolve, a topological distance on trees is defined in [19] 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 distance).

Definition 3 (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.*

For example, on figure 1.4, 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 ($f \rightarrow k \rightarrow j \rightarrow h \rightarrow g$), 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. The progressive neighbohood

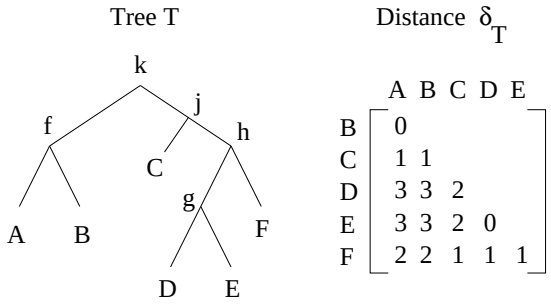


Figure 1.4: Example of topological distance δ_T

based on the topological distance was implemented in the software Hydra [19]. The reduction process used in Hydra takes into account a parameter M which corresponds to a maximum number of LS iterations. A parameter d is introduced to control the size of the neighborhood and is defined as the 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.

1.3.3.4 Other LS algorithms Several other LS algorithms can be used to solve the MP problem. Among them, we can distinguish:

Tabu Search (TS) [17] is a kind of descent-ascendant method for which a neighbor can be chosen to replace the current solution even if it does not improve the fitness function. To avoid the problem of possible cycling and to allow the search to overcome the local optimum problem, TS introduces the notion of Tabu list, a short term memory that maintains a selective history of previously encountered solutions. The size of the Tabu list tt, called Tabu tenure, prevents a solution to be reconsidered for the next tt iterations. [48] has applied TS to solve MP.

Simulated Annealing (SA) [27, 5] like TS accepts suboptimal solutions but with a certain probability. The principle of SA is inspired from annealing in metallurgy: a technique that involves heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. By analogy with this physical process, at each iteration the current solution is replaced by a random neighbor chosen with a probability that depends on the difference between the fitness function value and a global parameter T (called the temperature). The temperature is gradually decreased during the process such that the current solution changes almost randomly at the beginning of the search. The software LVB [4] is an application of SA to MP.

The GRASP metaheuristic (Greedy Randomized Adaptative Search Procedure) [10, 38] can be considered as an hybridization between a greedy construction method (here stepwise addition) and a LS mechanism. Every p iterations, GRASP reconsiders previous choices by improving the current (and incomplete) tree by Local Search. If $p = 1$, then a LS operation follows each taxon insertion. Generally, these LS stages are relatively short, and a complete LS is done at the end of the procedure (when all

the taxa are inserted). Moreover, LS stages allow to consider more randomness in the greedy part. [40] applied a GRASP+VNS heuristic and could obtain results of good quality on a set of benchmarks [2, 40].

1.3.4 Evolutionary metaheuristics and genetic algorithms

Evolutionary Algorithms (EA) [24, 31] represent another family of metaheuristics. Among them, *Genetic Algorithms* (GA) are based on a population of constant size of candidate solutions which undergo an evolution process with the use of operators named crossover, mutation and selection. The aim of the population is to explore different interesting areas of the search space in order to diversify the search. The crossover operators aims to create new candidate solutions (offspring) by combining two or more solutions (parents). The mutation operator locally alters offsprings by introducing randomness and consequently favoring diversity. A small number of LS steps with pure random selection (blind search) is a commonly used example of mutation. Finally, the selection operator determines which offspring will survive and reproduce. There exist different selection strategies like the roulette-wheel selection [3] or the tournament selection [20] for which the best individual is chosen after randomly picking n individuals. If an offspring survives it replaces one of the individuals of the population.

Algorithm 1.2

```
Genetic-Algorithm (S, f, N)
  P := { chose or generate n individuals ∈ S }
  for a given number of crossovers x do
    p, q := select-parents(P)
    r := crossover(p, q)                    // r is the offspring of p and q
    mutation(r)
    if selection(r) then
      replace(P, r)
    end if
  end for
```

1.3.4.1 GA related problems

It is now widely acknowledged that genetic operators like crossover and mutation should be tailored to the target problem in order to integrate problem-specific constraints and thus improve the search.

The literature describes several evolutionary algorithms for phylogenetic reconstruction: for instance [30, 28] for the Maximum Likelyhood problem, [33, 6, 7, 39] for the MP problem and [8] for distance-based phylogenetic approaches. Note that conventional subtree crossover operators used in tree-based genetic programming are not directly applicable here.

Most of these tree crossover operators follow the *subtree cutting and regrafting* strategy. More precisely, given two parents, a subtree is first selected from one parent (the donor). Then the leaves of this subtree are deleted from the other parent (the receiver), leading to an intermediate tree. The final child tree is obtained by regrafting the subtree from the donor on a merge point of the intermediate tree. Obviously, exchanging the donor and receiver allows to obtain a second child. Figure 1.5 shows

an example with 6 species, where the subtree (A, C) is removed from the donor and re-inserted in the receiver.

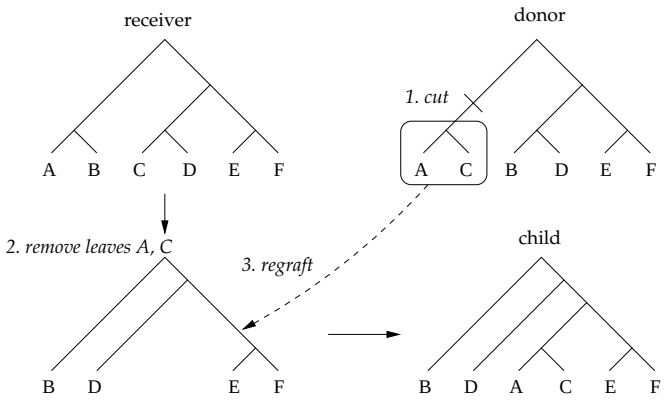


Figure 1.5: Cut and regraft example

One can observe that with such a crossover strategy, only partial information is transmitted from parents to offspring. In the example given in Figure 1.5, a subtree with 2 leaves (out of 6) of the donor tree is passed on to the child. In one sense, only a small portion of information of the donor is transmitted while a larger portion of information related to the 4 other species of the donor tree is lost during the crossover operation. To ensure a global combination and transmission of information during crossover operations, [18] introduce a specific operator based on the notion of *topological distance* between two leaves. The two parents are transformed into distance matrices, which are combined by using arithmetic operators in order to obtain an offspring matrix. From the offspring matrix a tree is generated using a distance method (for example UPGMA or NJ). This operator named DiBIP will be describe in the next section.

The Tree-Fusing [22] is an other example of crossover operation. Given two parent trees, this technique selects an improving tree among the population of valid trees constituted by exchanging subtrees between the parents.

Another important point to keep in mind is the diversification of the population. If the individuals are *too close* the search will only focus on one area of the search space. Generally, solutions of CO problems are represented as vectors of values, but in the case of MP the solutions are trees: while it is quite easy to compare vectors, it is less obvious to compare trees.

1.3.4.2 Distance-Based Information Preservation Crossover

The Distance-Based Information Preservation (DiBIP) crossover is based on the notion of *topological distance* (see definition 3) and aims to preserves *common* properties of parents in terms of topological distance between species. By common we mean that two

species that are close (resp. far) in both parents should stay close (resp. far) in the child.

The general approach (see Algorithm 1.3) can be summarized as follows: (1) calculate a distance matrix for each parent tree, (2) combine the matrices of the two parents to get a third matrix, and (3) create a child tree from this new matrix. T_1 and T_2 represent the input trees (parents), $\Delta : \mathcal{T} \rightarrow \mathcal{D}$ is a tree-to-distance operator that allows to obtain a distance matrix from a tree, $\oplus : \mathcal{D} \times \mathcal{D} \rightarrow \mathcal{D}$ is a matrix operator that allows to combine two distance matrices to produce a new distance matrix, and $\Lambda : \mathcal{D} \rightarrow \mathcal{T}$ is a distance-to-tree operator that allows to construct a tree given a distance matrix. The three operators Δ, $\oplus$ and Λ represent the three steps to transform two parent trees into one child which preserve topological properties shared by both parents.

Algorithm 1.3

function DiBIP(T_1, T_2, δ_T, Δ, $\oplus$, Λ)
D_1 = $\Delta(T_i)$ for (i=1,2)
D^*: $D^* = D_1 \oplus D_2$
$T^* = \Lambda(D^*)$
return T^*

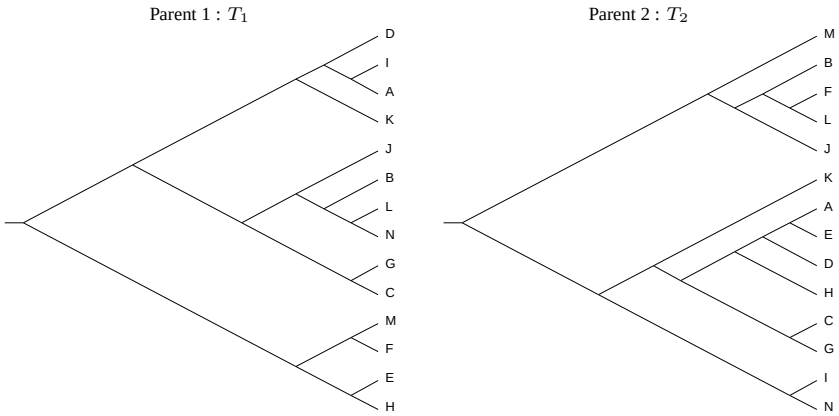


Figure 1.6: Example of input trees T_1 and T_2 for the DiBIP crossover

This general scheme gives rise to several comments: first, the distance measure should ideally be correlated to the evolutionary changes between species. For instance, two species separated by a small number of evolutionary changes should have a smaller distance than two species separated by a large number of changes. The *Hamming distance* is not appropriate here because this metric is totally independent of tree topologies.

Second, in order to preserve common properties of the parents during the crossover operation, a valid matrix operator $\oplus$ should meet some specific requirements meaningful to the MP problem in order to help transmit properties shared by both parents to

D_1	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	-	**B**												
B	6	-	**C**											
C	5	3	-	**D**										
D	1	5	4	-	**E**									
E	5	5	4	4	-	**F**								
F	5	5	4	4	2	-	**G**							
G	5	3	0	4	4	4	-	**H**						
H	5	5	4	4	0	2	4	-	**I**					
I	0	6	5	1	5	5	5	5	-	**J**				
J	5	1	2	4	4	4	2	4	5	-	**K**			
K	2	4	3	1	3	3	3	3	2	3	-	**L**		
L	7	1	4	6	6	6	4	6	7	2	5	-	**M**	
M	5	5	4	4	2	0	4	2	5	4	3	6	-	**N**
N	7	1	4	6	6	6	4	6	7	2	5	0	6	-

D_2	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	-	**B**												
B	8	-	**C**											
C	4	6	-	**D**										
D	1	7	3	-	**E**									
E	0	8	4	1	-	**F**								
F	9	1	7	8	9	-	**G**							
G	4	6	0	3	4	7	-	**H**						
H	2	6	2	1	2	7	2	-	**I**					
I	6	4	4	5	6	5	4	4	-	**J**				
J	7	1	5	6	7	2	5	5	3	-	**K**			
K	4	4	2	3	4	5	2	2	2	3	-	**L**		
L	9	1	7	8	9	0	7	7	5	2	5	-	**M**	
M	6	2	4	5	6	3	4	4	2	1	2	3	-	**N**
N	6	4	4	5	6	5	4	4	0	3	2	5	2	-

Figure 1.7: Topological distance matrices of T_1 and T_2

the child. For instance, if a pair of species (A, B) is closer than another pair (C, D) in both parents, then this property should be conserved by the crossover process and be transmitted to the resulting child. The arithmetic mean is a simple valid operator, even if it is possible to favor closeness or distant relations with the parametric operator given by $(D_1 \oplus D_2)(i,j) = \alpha.\min\{D_1(i,j), D_2(i,j)\} + (1-\alpha).\max\{D_1(i,j), D_2(i,j)\}$ with $\alpha \in [0,1]$.

While the resulting distance matrix is made with topological distances and not evolutionary ones, a simple clustering algorithm like UPGMA is well-adapted to provide a child tree. It ensures to aggregate taxa which are close in parents (depending on the selected $\oplus$ operator instantiation).

To illustratre this technique, we provide an example with two parents (see Fig. 1.6) for which we compute topological matrices D_1 and D_2 (see Fig. 1.7). For simplicity's

$$D^* = D_1 + D_2$$

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
A	-	**B**												
B	14	-	**C**											
C	9	9	-	**D**										
D	2	12	7	-	**E**									
E	5	13	8	5	-	**F**								
F	14	6	11	12	11	-	**G**							
G	9	9	0	7	8	11	-	**H**						
H	7	11	6	5	2	9	6	-	**I**					
I	6	10	9	6	11	10	9	9	-	**J**				
J	12	2	7	10	11	6	7	9	8	-	**K**			
K	6	8	5	4	7	8	5	5	4	6	-	**L**		
L	16	2	11	14	15	6	11	13	12	4	10	-	**M**	
M	11	7	8	9	8	3	8	6	7	5	5	9	-	**N**
N	13	5	8	11	12	11	8	10	7	5	7	5	8	-

Child : $T^* = \Lambda(D^*)$

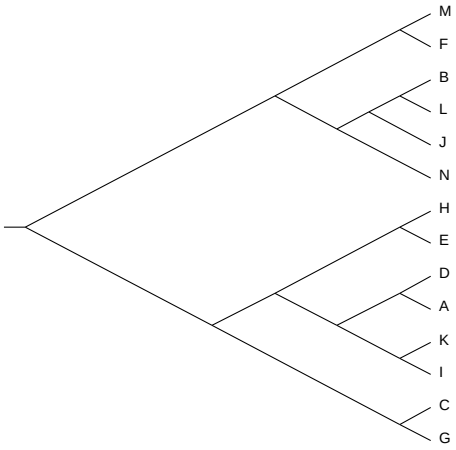


Figure 1.8: Topological distance matrix of child and tree that results from UPGMA for the DiBIP crossover

sake, we used the addition operator to combine the two parent matrices into the child matrix D^* (see Fig. 1.8).

1.3.5 Memetic methods

A memetic or hydrid algorithm (MA) is the combination of a GA helped by a LS improver [34]. Each time a new individual is generated by the GA, it is submitted

to a LS improvement. In algorithm 1.2, the mutation is then followed by a descent that starts with r. MA alternates intensification (Local Search) and diversification (crossover) stages. Despite the fact that MA are more computationally intensive they tend to provide solutions of better quality than GA.

1.3.6 Problem-specific improvements

Different techniques which are specific to MP have been developed in order to improve the search or decrease the computation time needed to obtain a tree of minimum score.

1.3.6.1 Divide and conquer methods For example, Sectorial Search (SS) [22] is a special kind of tree rearrangement that focuses on sectors (subtrees) of a tree. SS is based on the divide and conquer principle where subtrees can be analyzed more quickly than the overall tree. Sectors are analyzed separately and if a better configuration is found it will replace the previous one. Three different kinds of SS have been defined in [22]: *random* for which sectors are selected randomly, *consensus-based* and *mixed*. Experiments show that a sector should have between 35 and 55 nodes in order to obtain significant improvements. Another family of divide and conquer methods are the disc-covering methods (DCM) [35, 44]: DCM1 can be considered as an attempt to produce overlapping clusters to minimize the intracluster diameter and produces good subproblems. Nevertheless, the structure induced by the decomposition is often poor. DCM2 improves DCM1, but the subproblems obtained tend to be too large. Finally, DCM3 uses a dynamically updated guide tree in order to direct the decomposition. It then enables to focus the search on the *best* parts of the search space.

1.3.6.2 Multi-character optimization The most time consuming function in a search algorithm is the function of Fitch (see algorithm 1.4). This function takes as input two taxa $t1$ and $t2$. The output is the hypothetical taxon $t3$ and the number of changes returned by the function. Remember that the sum of all the changes of the overall tree constitutes the parsimony score.

Algorithm 1.4

```
function fitch(t1, t2, t3 : array[1..m] of char) : integer
  changes := 0
  for i := 1 to m do
    t3[i] := t1[i] ∩ t2[i]
    if (t3[i] == 0) then
      t3[i] := t1[i] ∪ t2[i]
      changes := changes + 1
    end if
  end for
  return changes
```

We can implement this function 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) unit which enables to *vectorize* the code. The vectorization of the code means the application of the same operation on different data at the same time. Intel and AMD processors offer a set of 8 SSE registers of 128 bits long on a 32 bits architecture. If we represent a nucleotide with one byte then a SSE register can store and handle 16 bytes (nucleotides) at a time.

In order to efficiently perform the union and intersection of algorithm 1.4, each character is represented by a power of 2, from $2^0 = 1$ ($-$) to $2^4 = 16$ (T), except for ? which can represent any other character and is then coded by the value $31 = 1 + 2 + \cdots + 16$. The union can be performed by the binary-OR (`|`) and the intersection by the binary-AND (`&`). The vectorization of Fitch's function gives a 90% improvement on Intel Core 2 Duo processors, while other architectures (pentium II/III/4, pentium-M, Athlon 64, Sempron) provide 70 to 80% improvement. This improvement then enables to divide the overall computation time of a program by a factor of 3 to 4. A first pseudo-code was given by [43] for PowerPC processors and recently [41] released the code for Intel and AMD processors. Finally, note that the most recent processors (Intel Core i5 or i7, AMD Phenom) introduce the SSE4.2 instruction set that contains the POPCNT function which counts the number of bits set to one in a register. This function is used essentially to determine the number of changes that occur when one performs the union between `t1[i:i+15]` and `t2[i:i+15]`. By replacing the implementation of POPCNT by the native SSE4.2 instruction we can see an overall improvement of 95% (on an Intel Core i7 860 processor) compared to the basic implementation.

1.3.6.3 Fast character optimization techniques

A set of methods [21, 16, 43, 47] 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 [21], Goloboff 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 [16] also proposed an algorithm which is exact and correct. In [47] a two passes algorithm is described which has the same complexity of Goloboff's and is faster than the incremental method of Gladstein.

1.4 CONCLUSION

Due to its inherent complex structure, the resolution of the large maximum parsimony problem can be efficiently achieved by means of optimization techniques.

Table 1.2 gives an overview of the complexity of the different methods described through out this chapter: n represents the number of taxa and m the number of sites

Method	Complexity
exhaustive	$O(n^n \times m)$
branch and bound	$O(n^n \times m)$
stepwise	$O(n^3 \times m)$ or $O(n^4 \times m)$
LS+NNI	$O(i \times n^2 \times m)$
LS+SPR	$O(i \times n^3 \times m)$
LS+TBR	$O(i \times n^4 \times m)$
GA	$O(p \times n \times m) + X \times (cross + mut + sel)$
MA	$O(p \times n \times m) + X \times (cross + mut + sel + LS)$

Table 1.2: Complexity of methods

of each taxa. For Local Search, i represents the number of iterations of the search. For Genetic Algorithms, p is the size of the population, X is the number of crossovers and $cross$, mut, sel are respectively the complexity of the crossover, mutation and selection operations. The last term LS is the complexity of the LS method used to improve a solution.

Local Search methods are the standard resolvers for the Maximum Parsimony problem and are widely used. Genetic Algorithms can obtain results of better quality than LS only if the crossover, mutation and selection operators are tailored to the problem. Finally, Memetic Algorithms, as a combination of GA and LS are the methods that can achieve results of very good quality but they sometimes lack efficiency compared to LS. A lot of other techniques like the Ratcheting technique or Sectorial Search are also very useful to escape from local optimum or to locally improve the overall tree.

New tools like the topological distance and the DiBiP crossover show that new approaches can help design *clever* techniques to help solve the MP problem more efficiently and reach solutions of higher quality.

References

1. B.L. Allen and M. Steel. Subtree transfer operations and their induced metrics on evolutionary trees. *Annals of Combinatorics*, 5(1):1–15, 2001.
2. A.A. Andreatta and C.C. Ribeiro. Heuristics for the phylogeny problem. *Journal of Heuristics*, 8:429–447, 2002.
3. J.E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms and their Application (ICGA2)*, pages 14–21, 1987.
4. D. Barker. Parsimony and simulated annealing in the search for phylogenetic trees. *Bioinformatics*, 20:274–275, 2004.
5. V. Cerny. A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.
6. C. B. Congdon. Gaphyl: An evolutionary algorithms approach for the study of natural evolution. *Proceedings of the 6th Joint Conference on Information Science*, 2002.
7. C. B. Congdon and K. J. Septor. Phylogenetic trees using evolutionary search: Initial progress in extending gaphyl to work with genetic data. *Proceedings of the 2003 Congress on Evolutionary Computation*, pages 320–326, 2003.
8. C. Cotta and P. Moscato. Inferring phylogenetic trees using evolutionary algorithms. *Lecture Notes in Computer Science*, 2439:720–729, 2002.
9. J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, 2003.

10. T. A. Feo and M. G. C. Resende. Greedy randomized adaptative search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

11. W. Fitch. Towards defining course of evolution: minimum change for a specified tree topology. *Systematic Zoology*, 20:406–416, 1971.

12. L.R. Foulds and R.L. Graham. The steiner problem in phylogeny is np-complete. *Advances in Applied Mathematics*, 3:43–49, 1982.

13. V. Ramachandran G. Ganapathy and T. Warnow. Better hill-climbing searches for parsimony. *Proceedings of the Third International Workshop on Algorithms in Bioinformatics (WABI)*, pages 245–258, 2003.

14. V. Ramachandran G. Ganapathy and T. Warnow. On contract-and-refine transformations between phylogenetic trees. *SODA*, pages 900–909, 2004.

15. M. R. Garey and D. S. Johnson. *Computers and intractability : a guide to the theory of NP-Completeness*. W. H. Freeman and Company, 1979.

16. D. S. Gladstein. Efficient character optimization. *Cladistics*, 13:21–26, 1997.

17. F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publisher, Boston, 1997.

18. A. Goëffon, J.M. Richer, and J.K. Hao. A distance-based information preservation tree crossover for the maximum parsimony problem. *Lecture Notes in Computer Science*, 4193:761–770, 2006.

19. 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.

20. D.E. Goldberg and K. Deb. *A comparative analysis of selection schemes used in genetic algorithms*, pages 69–93. Morgan Kaufmann Publishers, 1991.

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

22. P.A. Goloboff. Analyzing large data sets in reasonable times: solutions for composite optima. *Cladistics*, 15:415–428, 1999.

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

24. J.H. Holland. *Adaptation in natural and artificial systems*. The University of Michigan Press, 1975.

25. H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, 2004.

26. I. Horovitz. A report on one day symposium on numerical cladistics. *Cladistics*, 15:177–182, 1999.

27. S. Kirkpatrick, C. Gellat, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

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

29. H. R. Lourenço, O. Martin, and T. Stützle. *Iterated Local Search*. Handbook of Metaheuristics. Kluwer Academic Publisher, Boston, 2002.

30. H. Matsuda. Protein phylogenetic inference using maximum likelihood with a genetic algorithm. *Prof. of Pacific Symposium on Biocomputing*, pages 512–536, 1996.

31. M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1998.

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

33. A. Moilanen. Searching for most parsimonious trees with simulated evolutionary optimization. *Cladistics*, 15(3):39–50, 1998.

34. P. Moscato. *Chapter Memetic Algorithms : A Short Introduction in New Ideas in Optimization*. McGraw-Hill, 1999.

35. L. Nakhleh, U. Roshan, K. St John, J. Sun, and T. Warnow. Designing fast converging phylogenetic methods. *Bioinformatics Supplement*, 17:190–198, 2001.

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

37. 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.

38. M. G. C. Resende and C. C. Ribeiro. *Greedy randomized adaptative search procedures*. In Handbook of Metaheuristics. F. Glover abd G. Kochenberger editors, 2003.

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

40. 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*, 12:1–14, 2005.

41. J.-M. Richer. Three new techniques to improve phylogenetic reconstruction with maximum parsimony. Technical report, LERIA, 2008.

42. D.F. Robinson. Comparison of labeled trees with valency three. *J. Combin. Theory*, 11:105–119, 1971.

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

44. U. Roshan, B.M.E. Moret, T.L. Williams, and T. Warnow. Rec-i-dcm3: A fast algorithmic technique for reconstructing large phylogenetic trees. *Proceedins of IEEE Computational Systems Bioinformatics Conference (CSB 04)*, pages 98–109, 2004.

45. D.L. Swofford and G.J. Olsen. *Molecular Systematics*. D.M. Hillis and C. Moritz, 1990.

46. M.S. Waterman and T.F. Smith. On the similarity of dendograms. *Journal of Theoretical Biology*, 73:789–800, 1978.

47. 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.

48. L. Yu-Min, F. Shu-Cherng, and T. L. Jeffrey. A tabu search algorithm for maximum parsimony phylogeny inference. *European Journal of Operational Research*, 176(3):1908–1917, February 2007.