ABSTRACT

LIN, YU-MIN. Tabu Search and Genetic Algorithms for Phylogeny Inference. (Under the direction of Professor Shu-Cherng Fang).

Phylogenetics is the study of evolutionary relations between different organisms. Phylogenetic trees are the representations of these relations. Researchers have been working on finding fast and systematic approaches to reconstruct phylogenetic trees from observed data for over 40 years. It has been shown that, given a certain criterion to evaluate each tree, finding the best fitted phylogenetic trees among all possible trees is an NP-hard problem [27]. In this study, we focus on the topology searching techniques for the maximum-parsimony and maximum-likelihood phylogeny inference. We proposed two search methods based on tabu search and genetic algorithms. We first explore the feasibility of using tabu search for finding the maximum-parsimony trees. The performance of the proposed algorithm is evaluated based on its efficiency and accuracy. Then we proposed a hybrid method of the tabu search and genetic algorithm. The experimental results indicate that the hybrid method can provide maximum-parsimony trees with a good level of accuracy and efficiency. The hybrid method is also implemented for finding maximum-likelihood trees. The experimental results show that the proposed hybrid method produce better maximum-likelihood trees than the default-setting dnaml program in average on the tested data sets. On a much larger data set, the hybrid method outperforms the default-setting dnaml program and has equally good performance as the dnaml program with the selected jumble option.

Tabu Search and Genetic Algorithms for Phylogeny Inference

by
Yu-Min Lin

A dissertation submitted to the Graduate Faculty of
North Carolina State University
in partial fullfillment of the
requirements for the Degree of
Doctor of Philosophy

Operations Research

Raleigh, North Carolina

2008

APPROVED BY:

Dr. Steffen Heber	Dr. Henry L. W. Nuttle
Dr. Shu-Cherng Fang Chair of Advisory Committee	Dr. Jeffrey L. Thorne Co-Chair of Advisory Committee

DEDICATION

I would like to dedicate this to my parents and my wife, whose unwavering support and encouragement helped me survive this journey. I would also like to dedicate this to my daughter, who gives me the courage and motivation to finish my work in time.

BIOGRAPHY

Yu-Min Lin, son of Song-Cherng Lin and Kuei-Chun Wu, was born on 29 July, 1976 in Taipei, Taiwan. He was raised in Taipei, where his parents own a family business. Yu-Min graduated from National Taiwan University with a bachelor degree of Business Administration in 2002. Upon graduation, he came to North Carolina State University and initiated his master's program in Operations Research in August, 2002. After passing the qualify exam in August, 2003, he initiated his PhD program in Operations Research in Spring, 2004.

ACKNOWLEDGMENTS

This work would not be completed without the kind assistance of the people behind the study. I wish to convey my sincere gratitude to the following:

To Dr. Shu-Cherng Fang, my advisor, for his support and guidance to my research and my life over the last several years.

To Dr. Jeffrey Thorne, for giving me great help and valuable advises to improve my work.

To Dr. Steffen Heber and Dr. Henry Nuttle, for their effort in going through the progress of my research project

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

Chapter 1

Introduction

In this chapter, an introduction of phylogeny inference will be given in section 1.1. Four widely-used tree-building methods for inferring phylogeny are described in section 1.2. The objective of our research and an outline of this dissertation can be found in section 1.3 and 1.4, respectively.

1.1 Phylogeny Inference

The observations of similarity among organisms strongly suggest that all organisms on earth may share a common ancestor. The evolutionary development and history of a taxonomic grouping of organisms is called phylogeny. In the study of phylogeny, evolution is considered as a branching process. During this process, population of species may speciate into different branches, hybridize together or terminate by extinction. Hence, a phylogenetic tree (e.g., Fig. 1.1) is usually used to represent the relationship among different species and the history of evolution.

Before the rapid development of molecular biology, the major means to infer phylogeny were morphology, physiology, anatomy and paleontology. With the advances of molecular sequencing technology, large amounts of molecular data, such as DNA sequences and amino acid sequences, can now be used for phylogeny inference. There are several reasons why molecular data are more suitable than traditional means for phylogenetic studies. First, the description of molecular characters is less ambiguous than morphological or physiological data. Second, the molecular traits generally evolve in a more regular

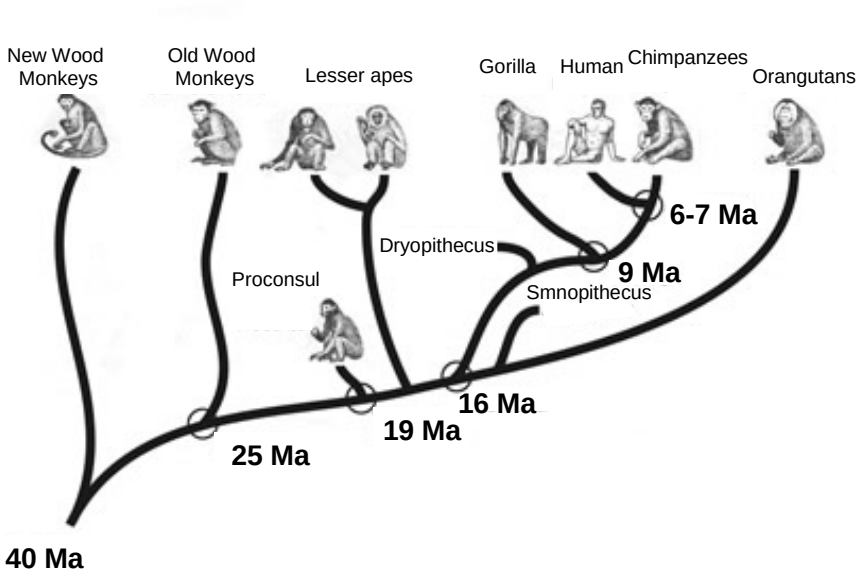


Figure 1.1: An example of phylogenetic tree for primates. 1 Ma = one million years.
Modified from http://www.news.utoronto.ca/bios/askus4.htm

manner. Third, the evolutionary relationship among very distantly related organisms can be inferred using molecular data. Fourth, molecular data are more manageable for the quantitative treatments. However, compared to paleontology where the fossil data can be used to infer the evolutionary relationship among prehistoric species, one issue posed by molecular phylogeny is that the molecular data are only available for the extant organisms. The hypothesized states of ancestral species need to be inferred based on our evolutionary knowledge.

Based on different evolutionary assumptions, various phylogenetic tree-building methods have been built to infer molecular phylogeny. Although these methods may differ in many ways, the common goal is to find phylogenetic trees that have the best explanation for the similarities and differences between sequences of different organisms. In the following section, we will briefly describe four widely-used methods for inferring phylogeny.

1.2 Phylogenetic Tree-building Methods

1.2.1 Distance-based Methods

The family of distance-based (or distance matrix methods) was introduced by Cavalli-Sforza and Edwards [6] and Fitch and Margoliash [21], while the idea of using pairwise distance between species was first proposed by Sokal and Michener in 1958 [52]. The general idea is to calculate a measure of the distance between each pair of species for finding a tree that predicts the observed set of distances as closely as possible. In distance matrix methods, we consider the distances as estimates of branch length separating the pair of species. The branch lengths here are not simply a function of time. They reflect expected amounts of evolution in different branches of the tree. Two branches may reflect the same elapsed time but might have different expected amounts of evolution. In a more general case, we can even allow different branches to have different rates of evolution.

The *least square methods* are some of the best-justified distance-based methods from a statistical point of view. Suppose we have an observed matrix of distances (D_{ij}), which is derived from sequence data or other observed characters. For any given tree topology and branch length set, we can also compute a predicted distance matrix (d_{ij}). The way to derive the predicted distance matrix is illustrated in Fig. 1.2.

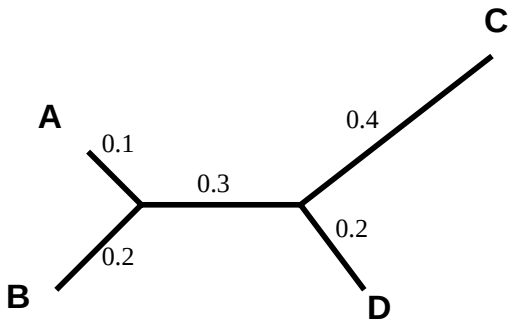
Then we can define a measure of the discrepancy between the observed and the expected distances as

$$Q = \sum_{i=1}^{n} \sum_{j=1}^{n} w_{ij}(D_{ij} - d_{ij})^2, \tag{1.1}$$

where w_{ij} are weights that differ for different least square methods. Since the predicted distances are a linear combination of branch lengths, we can optimize the branch lengths by taking the derivative of Equation (1.1) with respect to the branch lengths and equating it to zero for any given tree topology.

Being able to assign branch lengths to each tree topology, we can search among all tree topologies to find least square trees. However, this is challenging because the number of possible tree topologies is huge and finding a least squares tree is an NP-complete problem [11].

Another class of distance matrix methods applies clustering algorithms to the



	A	B	C	D
A	0	0.3	0.8	0.6
B	0.3	0	0.9	0.7
C	0.8	0.9	0	0.6
D	0.6	0.7	0.6	0

Figure 1.2: A tree and the predicted distance matrix, which are calculated by adding up the lengths of branches between each pair of species.

distance matrix to come up with a tree directly. The most popular ones among them are the *UPGMA* method [52] and the *neighbor-joining* (NJ) method [49].

In the UPGMA method, we assume the branch lengths satisfy a molecular clock. Clocklike trees are rooted, and the total path length from the root to a leaf is assumed to be equal to the total path length from the root to any other leaf. These kinds of trees are often referred to as being ultrametric. Although UPGMA is a very quick and simple algorithm to find a tree, it may give us seriously misleading results if the distances actually reflect a substantially non-clocklike tree (Fig. 1.3).

The neighbor-joining method is an algorithm that works by clustering. Unlike UPGMA, it does not assume a clock but, instead, approximates the minimum evolution method. The approximation is in fact quite good. It is guaranteed to recover the true tree if the distance matrix happens to be an exact reflection of a tree. Generally speaking, both UPGMA and neighbor-joining methods are very efficient. They can practically deal with more than hundreds of species [13].

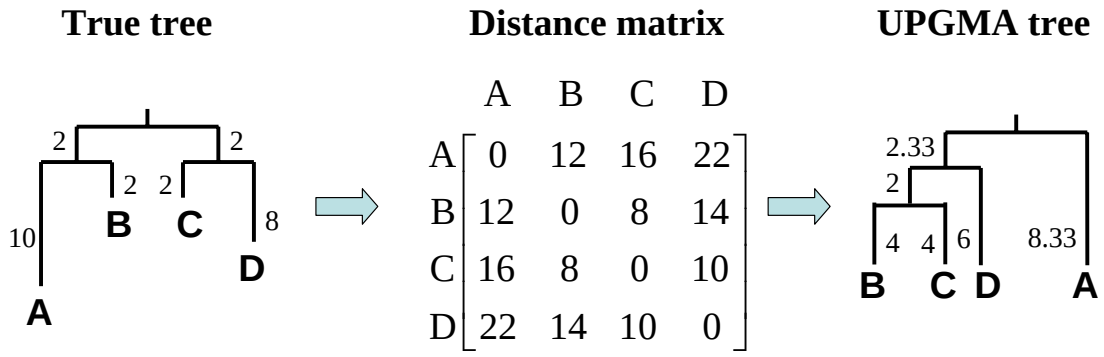


Figure 1.3: A four-leaf, nonclocklike tree and its corresponding distance matrix. The tree estimated by applying the UPGMA method to this distance matrix is also shown above. We can find that it does not have the correct tree topology.

1.2.2 Maximum Parsimony Methods

Maximum parsimony methods (or parsimony methods) were among the first methods for inferring phylogenies. They probably are the easiest ones to explain. The general idea of parsimony methods is that a particular phylogenetic tree is preferred if this tree can be explained with the minimum amount of net evolutionary changes [14]. Therefore, the object of parsimony methods is to seek phylogenetic trees that involve as few evolutionary changes as possible. This raises two issues: (i) how to make the reconstruction of evolutionary changes, involving as few changes as possible, for a proposed phylogenetic tree and (ii) how to find the topologies that minimize the number of changes among all possible phylogenetic trees.

Many methods have been developed to deal with the first issue. We will discuss here two algorithms that generalize these methods. The first one was proposed by Fitch [19] and the second by Sankoff [51]. An example of the *Fitch algorithm* is shown in Figure 1.4.

The Fitch algorithm considers the sites (or characters) one at a time. Then the number of changes is added up for all sites. At each leaf of a tree, we create a set that contains those observed characters (nucleotides or amino acids, for example). Hence, if we see a C in a leaf, we create the set $\{C\}$. Then we move from the exterior nodes to the root

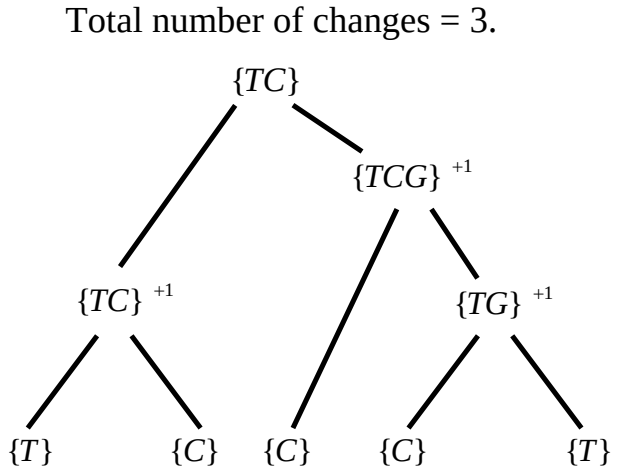


Figure 1.4: An example of counting evolutionary changes by using Fitch algorithm.

of the tree. At each interior node we create a set that is the intersection of sets at the two descendant nodes (or child nodes from here on). However, if the set is empty, we instead create the set that is the union of the two sets at the child nodes and increase the total number of changes by one. As shown in Fig. 1.4, we know that 3 is the total number of changes by applying the Fitch algorithm.

The Fitch algorithm is both effective and efficient. But it does not tell us what to do if we want to count different kinds of changes separately. The *Sankoff algorithm* is more flexible but complex in counting changes. Assume that we have a cost matrix $[c_{ij}]$ that represents the cost of change between each character state. More specifically, let c_{ij} represent the cost of change from character i to character j, then c_{ij} is not necessarily equal to c_{ji}. Also, let $S_k(i)$ denote the minimum cost at node k given that node k is assigned character i. Suppose ℓ and r are two child nodes of node k, then $S_k(i)$ can be defined as

$$S_k(i) = \min_j \{c_{ij} + S_\ell(j)\} + \min_p \{c_{ip} + S_r(p)\}. \tag{1.2}$$

If node k is an exterior node, then $S_k(i)$ will be 0 if the observed character is i, and infinite otherwise. As before, we compute the cost of each node by moving from leaves up to the root. Suppose $S_0(i)$ represents the cost at the root node given that the root node is assigned

character i. Then the minimum cost of this tree is given by

$$S = \min_i \{ S_0(i) \} \tag{1.3}$$

An example of the Sankoff algorithm is demonstrated in Fig. 1.5.

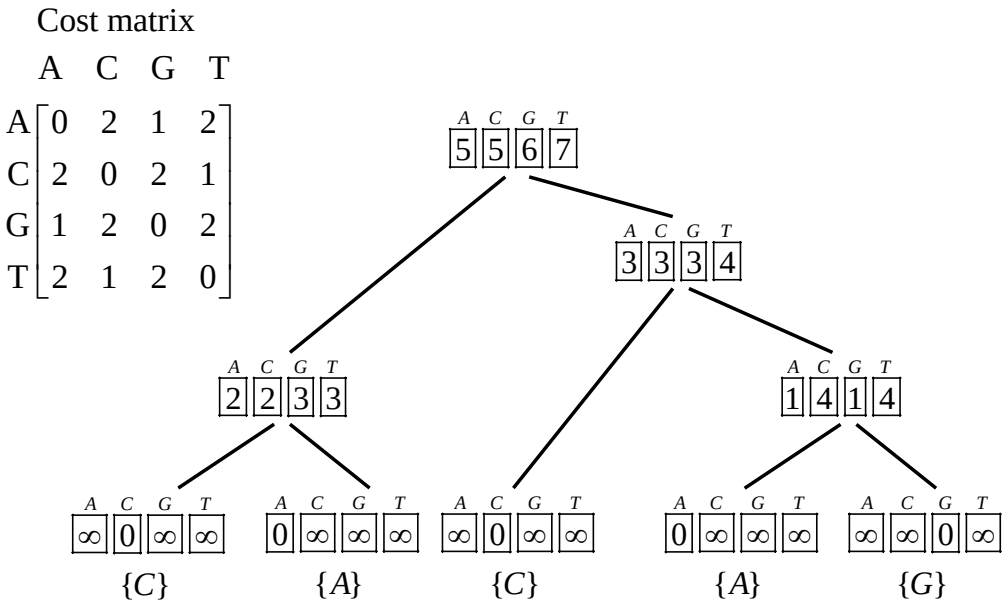


Figure 1.5: An example of Sankoff algorithm. The cost matrix used is shown, as well as the cost array at each node of the tree. The minimum cost of the tree is 5.

Once we can count the evolutionary changes for a given tree, the remaining problem is how to find a tree that involves a minimum number of evolutionary changes. This is as difficult as finding the least squares trees. Foulds and Graham proved that finding the most parsimonious tree (or trees) is NP-hard [27]. Many heuristic algorithms have been developed to improve the searching efficiency and some of them will be introduced in the later section.

1.2.3 Maximum Likelihood Methods

Maximum likelihood is of central importance in statistics and perhaps the most standard statistical method to make estimates of the phylogeny. The first likelihood method for phylogenies was introduced by Edwards and Cavalli-Sforza [15]. Felsenstein proposed

an algorithm, called *pruning algorithm*, to conduct likelihood computation more efficiently [16, 17].

In order to understand the calculation of likelihood, we first examine a simple example shown below:

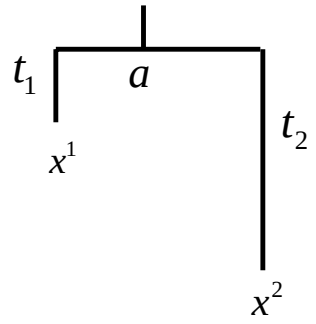


Figure 1.6: A simple tree.

x^1 and x^2 denote two observed sequences. The branch lengths from the root to these two leaves are t_1 and t_2, respectively. Consider only a site u. We denote the residues at these two leaves x_u^1 and x_u^2, respectively. We also assign a residue a to the root. Then, the likelihood of this simple tree at site u becomes

$$P(x_u^1, x_u^2, a|T, t_1, t_2) = \pi_a P(x_u^1|a, t_1) P(x_u^2|a, t_2), \tag{1.4}$$

where T denotes the tree topology and π_a is the equilibrium probability of drawing a from the distribution of character states at root. $P(x_u^i|a, t_i)$ is the transition probability of node x^i having residue x_u^i given the residue of its preceding node is a and the branch length is t_i. Since in general we do not know what the root residue was, we have to sum over all possible character states to get the probability of x_u^1 and x_u^2:

$$P(x_u^1, x_u^2|T, t_1, t_2) = \sum_a \pi_a P(x_u^1|a, t_1) P(x_u^2|a, t_2). \tag{1.5}$$

If there are N sites, then we can write the full likelihood of this simple tree as

$$P(x^1, x^2|T, t_1, t_2) = \prod_{u=1}^{N} P(x_u^1, x_u^2|T, t_1, t_2). \tag{1.6}$$

The computation of (1.6) cannot be completed without finding the transition probability $P(x_u^i|a, t_i)$. Assume the transition process be Markovian and stationary. Then, after

defining the transition rates between each state, the transition probability can be obtained. The Jukes-Cantor model (see Fig. 1.7) is the simplest one of the transition rates for DNA sequence data [33]. The transition probability $P(A|C,t)$, for example, can be evaluated as

$$P(A|C,t) = \frac{1}{4}(1 - e^{-\frac{4}{3}\gamma t}) \tag{1.7}$$

Note that Jukes-Cantor's model is also a time-reversible model, which means the probability of starting from state 1, and evolving to state 2, is the same as the probability of starting from state 2 and evolving to state 1.

To extend the above procedure to a tree with more sequences, Felsenstein introduced his *pruning algorithm*, a dynamic-programming-based algorithm, to efficiently compute the likelihood of a multi-species tree [16]. The detail of this algorithm will be presented in Chapter 7.

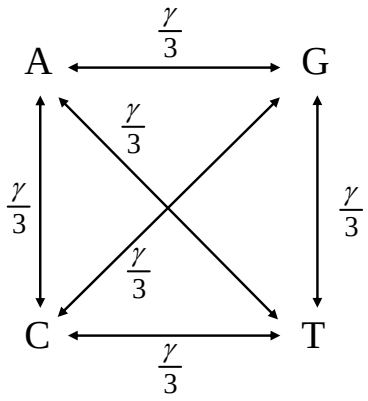


Figure 1.7: Jukes-Cantor model of transition rates on DNA data.

1.2.4 Bayesian Methods

Bayesian methods are closely related to maximum likelihood methods. The main difference is the use of a prior distribution of the quantity being inferred, which would be the tree in our case. The basic idea of Bayesian inferences goes as below. Given a hypothesis tree H and the observed data D, the posterior probability is

$$P(H|D) = \frac{P(H,D)}{P(D)} = \frac{P(H)P(D|H)}{\sum\limits_{H} P(H)P(D|H)}. \tag{1.8}$$

Notice that $P(D|H)$ is the likelihood as mentioned in the previous subsection. In Bayesian inference, the denominator in Eq. (1.8) is very difficult to compute, since it involves summing over all possible hypotheses (including all possible tree topologies and all possible branch lengths). Fortunately, the *Markov Chain Monte Carlo* (MCMC) method provides a way to draw samples from the posterior distribution without the need of knowing the denominator.

One of the most widely used Markov Chain Monte Carlo methods is the Metropolis algorithm [42]. This algorithm involving the following steps [17]:

1. Start at some tree, called T_i.
2. Pick a tree, say T_j, from the neighborhood of T_i.
3. Compute the ratio of the probabilities of the proposed new tree and the old tree:
$$R = \frac{P(T_j)P(D|T_j)}{P(T_i)P(D|T_i)}. \tag{1.9}$$
4. If $R \geq 1$, accept the new tree as the current tree.
5. If $R < 1$, draw a random number $x \in [0, 1]$. If $x < R$, accept the new tree as the current tree.
6. Return to step 2.

Note that the denominator in (1.8) is cancelled out in the expression (1.9) for the acceptance ratio. Also note that there is no terminating condition in this algorithm. In practical, this algorithm will be terminated whenever we collect enough samples for estimating the posterior distribution. This process is a Markov chain because it is a random process in which the next change depends only on the current state and not on where the process was previously.

The acceptance ratio is the ratio of the prior probabilities of the proposed and current trees, multiplied by the likelihood ratio of these two trees. We already mentioned how to compute the likelihood of a tree. Then one needs to choose a prior distribution for which the probabilities can be computed. Nevertheless, this is a controversial issue in Bayesian inference. Researchers doing a Bayesian analysis of a data set apply a prior at their own choice. There is no universal prior that is agreed by all to be a valid one. In fact,

the issue of whether the prior exists or not becomes a controversial issue in the philosophy of science.

Another issue is how to summarize the posterior after having samples from the Metropolis algorithm. This requires the information of how frequent each tree is. If we consider the branch length of a tree as a continuous time interval, the probability of a tree being selected is close to zero. If we consider the tree topologies only, the probabilities may still be hard to get assessed. One solution is to use clade probabilities. A clade can be considered as a group of species which descend from a common ancestor. For example, if we are interested in clades such as {Human, Chimp}, then we can sum the posterior probabilities of all trees that contain this clade and say that the probability of this clade in the tree is 0.92 (for example). More details about the clade probabilities can be found in Larget and Simon's paper [35].

1.3 Scope and Objective of Research

In this research, we focus on the phylogeny inference with maximum-parsimony and maximum-likelihood methods. As described in the previous section, the mechanisms of these two methods are similar: search through all possible tree topologies and find the ones that have the maximum-parsimony or maximum-likelihood values. Therefore, they can be viewed as optimization problems. Various heuristic search strategies have been proposed to improve the efficiency of finding the maximum-parsimony or maximum-likelihood trees. However, there are very few studies that explore the feasibility of using tabu search on the maximum-parsimony and maximum-likelihood phylogeny inference problems.

The major objective of this study is to incorporate the tabu search for finding maximum-parsimony and maximum-likelihood trees. We propose a topology search algorithm based on tabu search for the maximum-parsimony phylogeny inference. We also propose two hybrid methods, which combine genetic algorithm and tabu search, for the maximum-parsimony and maximum-likelihood phylogeny inference, respectively. The performance of the proposed algorithms will be evaluated.

1.4 Organization of Dissertation

Chapter 2 describes the structure of tree topologies. A survey of previous studies of heuristic searches for maximum-parsimony and maximum-likelihood phylogeny inference is given in Chapter 2. In Chapter 3, we present a phylogenetic tabu search for maximum-parsimony inference [38]. Its performance will be evaluated in Chapter 4. We present a hybrid heuristic search combining genetic algorithm and tabu search for finding maximum-parsimony phylogenetic trees in Chapter 5. The performance of the hybrid method will be evaluated in Chapter 6. In Chapter 7, the proposed hybrid method is modified for finding the maximum-likelihood phylogenetic trees. Its performance is assessed in Chapter 8. A summary of our research and suggestions for future study are presented in Chapter 9.

Chapter 2

Problem Description

We give a detailed description of the maximum-parsimony and maximum-likelihood phylogeny inference problems in this chapter. Section 2.1 provides the anatomy of phylogenetic trees. In Section 2.2, optimization models are formulated for both of the maximum-parsimony and maximum-likelihood phylogeny inference problems. Section 2.3 provides several commonly-used tree reconstruction techniques. A review of known topology searching methods for finding the maximum-parsimony and maximum-likelihood trees is given in Section 2.4.

2.1 Anatomy of Phylogenetic Trees

A phylogenetic tree is a graph composed of nodes and branches. A node represents a taxonomic unit, such as populations, species, individuals, or genes. The branches define the relationships among the taxonomic units. The branching pattern of a phylogenetic tree is called a topology. As shown in Fig. 2.1, the nodes locate at the terminal end of the tree are called *exterior nodes* (or *leaves*), while all others are called *interior nodes*. Exterior nodes represent the existing taxonomic units under comparison, and interior nodes represent the inferred ancestral units (or hypothetical taxonomic units). A node is *bifurcating* if it connects to only two descendant branches. If a node connects to more than two descending branches, it is *multifurcating*. For example, nodes G and H in Fig. 2.1 are bifurcating while node F is multifurcating. If all the interior nodes of a tree are bifurcating, this tree is called a bifurcating tree or binary tree. If a tree contains at least one multifurcating node,

it is called a multifurcating tree. In our research, all phylogenetic trees are presented as bifurcating trees since any multifurcating tree can be converted to a bifurcating one [20, 30].

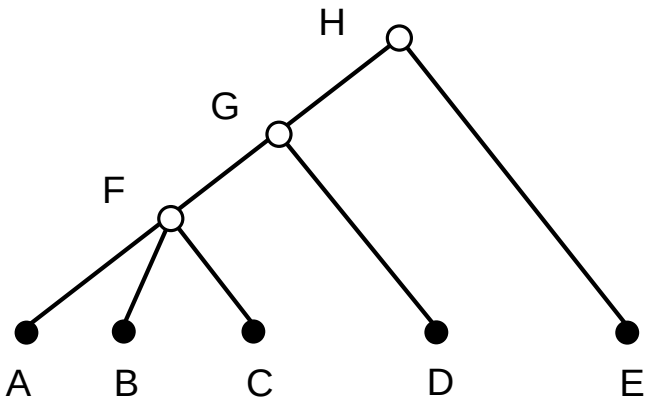


Figure 2.1: An example of a phylogenetic tree. Nodes A, B, C, D and E are the exterior nodes. Nodes F, G and H are interior nodes.

The phylogenetic trees can also be classified into *rooted trees* and *unrooted trees* (see Fig. 2.2). The biological interpretation of a root is that it represents the most recent common ancestor of the leaves on the tree. One can observe that the rooted tree (b) of Fig. 2.2 is generated from the unrooted tree (a) by placing a root on branch r. The rooted tree indicates a unique evolutionary path from the root to the exterior nodes while the unrooted tree only specifies the relationship among the exterior nodes and does not define any evolutionary path.

In Fig. 2.2 (a), the unrooted tree with 4 leaves can be converted to a rooted tree by placing the root on any of the 5 branches. In general, there are $2k-3$ branches in a k-leaf unrooted tree and therefore there are $2k-3$ rooted trees corresponding to the same unrooted tree. Suppose we have an unrooted tree with $k-1$ leaves, then the number of branches on which we can place an additional leaf is $2k-5$. Therefore, by sequentially adding leaves into a tree, it is not hard to show that the number of unrooted trees with n leaves is

$$1 \times 3 \times 5 \times \cdots \times (2n-5). \tag{2.1}$$

Since there are $2n-3$ rooted trees corresponding to the same unrooted tree, the number of

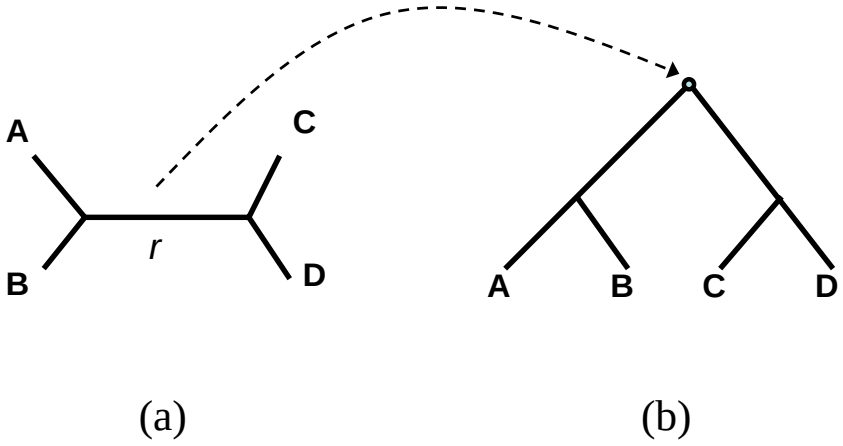


Figure 2.2: Unrooted tree(a) versus rooted tree (b).

rooted trees with n leaves becomes

$$1 \times 3 \times 5 \times \cdots \times (2n-3). \tag{2.2}$$

Table 2.1 provides the number of possible rooted and unrooted trees for different number of leaves on trees. When the number of leaves on a tree is small, the inference of tree topologies can be done by enumeration. However, the number of trees drastically increases when the number of leaves increases. Exhaustive enumeration is then impractical when the size of tree becomes large.

2.2 Problem Formulations

Assume that the graph $G(\mathscr{N}, \mathscr{B})$ is a tree topology consisting of node set $\mathscr{N}$ and the corresponding branch set $\mathscr{B}$. Let $\mathscr{F}$ be a function which evaluates the evolutionary changes (or parsimony score) for any given $G(\mathscr{N}, \mathscr{B}) \in \{binary\ trees\}$. The problem of finding the maximum-parsimony trees can be formulated as

$$\begin{aligned} \min \quad & \mathscr{F}\big(G(\mathscr{N}, \mathscr{B})\big) \\ \text{subject to} \quad & G(\mathscr{N}, \mathscr{B}) \in \{binary\ trees\}. \end{aligned} \tag{2.3}$$

Notice that the branch lengths are not included in this formulation since they do not affect the parsimony score of a tree. Also, a global optimal solution of Eq. (2.3) exists since there are a finite number of feasible solutions.

Table 2.1: Number of possible rooted trees and unrooted trees with increasing number of leaves.

Number of leaves	Number of rooted trees	Number of unrooted trees
2	1	1
3	3	1
4	14	3
5	105	14
6	945	105
7	10395	945
8	135135	10395
9	2027025	135135
10	34459425	2027025
11	654729075	34459425
12	13749310575	654729075
13	316234143225	13749310575
14	7905853580625	316234143225
15	213458046676875	7905853580625
16	6190283353629375	213458046676875
17	191898783962510625	6190283353629375
18	6332659870762850625	191898783962510625
19	221643095476699771875	6332659870762850625
20	8200794532637891559375	221643095476699771875
30	4.9518×10^{38}	8.6874×10^{36}
40	1.0099×10^{57}	1.3115×10^{55}

Let Ω be the parameter set of a evolution model, $\ell_{\mathscr{B}}$ be the branch lengths of the corresponding branch set $\mathscr{B}$ and $G(\mathscr{N}, \mathscr{B}, \ell_{\mathscr{B}})$ be a tree composed of node set $\mathscr{N}$, branch set $\mathscr{B}$ and the corresponding branch lengths $\ell_{\mathscr{B}}$. Let $\mathscr{L}$ be a function which evaluates the likelihood for any given $G(\mathscr{N}, \mathscr{B}, \ell_{\mathscr{B}})$. Then, the problem of finding the maximum-likelihood trees can be formulated as

$$\begin{aligned} \max \quad & \mathscr{L}\big(G(\mathscr{N}, \mathscr{B}, \ell_{\mathscr{B}}), \Omega\big) \\ \text{subject to} \quad & G(\mathscr{N}, \mathscr{B}) \in \{\textit{binary trees}\}, \\ & \ell_{\mathscr{B}} \geq 0. \end{aligned} \tag{2.4}$$

Note that this optimization problem is bounded above since the largest possible value of $\mathscr{L}$ function is 1.0. For the computational purpose, the objective function is usually changed to the log-likelihood form. This change will not affect us since the logarithm function is monotonically increasing. Assume that we can find optimal branch lengths, $\ell^*_{\mathscr{B}}$, for any given $G(\mathscr{N}, \mathscr{B}, \ell_{\mathscr{B}})$, then the global optimal solutions of Eq. (2.4) should exist and be achievable since the number of solutions $G(\mathscr{N}, \mathscr{B}, \ell^*_{\mathscr{B}})$ is finite.

As mentioned in Section 1.2.2 and 1.2.3, both of the problems are NP-hard. In several recent studies, problem of (2.3) is converted to an integer-programming (IP) problem or mixed-integer-linear-programming (MILP) problem [5, 29, 53]. The IP and MILP problems are then solved in near polynomial time for *perfect phylogeny* or *near-perfect phylogeny*. However, there are some limitations of these methods. First, these methods are applied to solving the haplotype phylogeny inference problems only. The absence of IP or MILP formulations for a general phylogeny inference problem remains to be challenging. Second, the size of the IP or MILP models may be, in the worst case, doubly-exponential in the number of leaves of the tree. Hence, we focus on developing new approaches for solving the maximum-parsimony and maximum-likelihood phylogeny inference problems in this research.

A commonly-used platform for solving our problems involves the following steps:

1. Generate initial feasible solution(s).
2. If a given stopping condition is met, STOP. Otherwise, go to next step.
3. Reconstruct new feasible solutions by modifying current solution(s).
4. Update current solution(s). Go to step 2.

This platform works for the so-called topology searching methods. The success of a topology search method requires an efficient search strategy and a good technique to generate and reconstruct phylogenetic trees. We will introduce some widely-used techniques to generate the initial trees and reconstruct new trees by modification.

2.3 Generate and Reconstruct Phylogenetic Trees

2.3.1 Generate phylogenetic trees

Most of the techniques for generating the initial trees can be classified into two categories: *sequential addition* and *star decomposition*. Also, Fig. 2.3 gives an example of generating a 5-leaf tree by using sequential addition and star decomposition. Sequential addition starts from a randomly generated 3-leaf tree. The rest of the species are randomly selected one-by-one. A selected species is inserted to the tree by attaching on one of the branches of the tree. This process continues until all species are inserted to the tree. Star decomposition operates in the opposite way. It starts with all species attaching on the tree but unresolved, i.e., no grouping pattern is formed. The tree is then gradually resolved by grouping two branches at a time. This process continues until the tree is fully resolved with no multifurcating node left.

Both sequential addition and star decomposition can be modified to generate trees with better quality. For sequential addition, the selected species is inserted to every branch of the tree. Compute the objective values (parsimony score for maximum-parsimony phylogeny and likelihood score for maximum-likelihood phylogeny) of the trees after the insertion. Keep the best one for the next insertion and discard the others. For star decomposition, the "distances" between each pair of species are computed (see Section 1.2.1). Then the *neighbor-joining* or *UPGMA* method can be applied to group branches together.

2.3.2 Reconstruct Phylogenetic Trees

In this subsection, we introduce four widely-used techniques to rearrange phylogenetic trees when doing a topology search. They are *subtree pruning and regrafting* (SPR), *tree bisection and reconnection* (TBR), *nearest-neighbor interchanges* (NNI), and *tree-fusing*. They are also referred to as *tree rearrangement* techniques. We only demon-

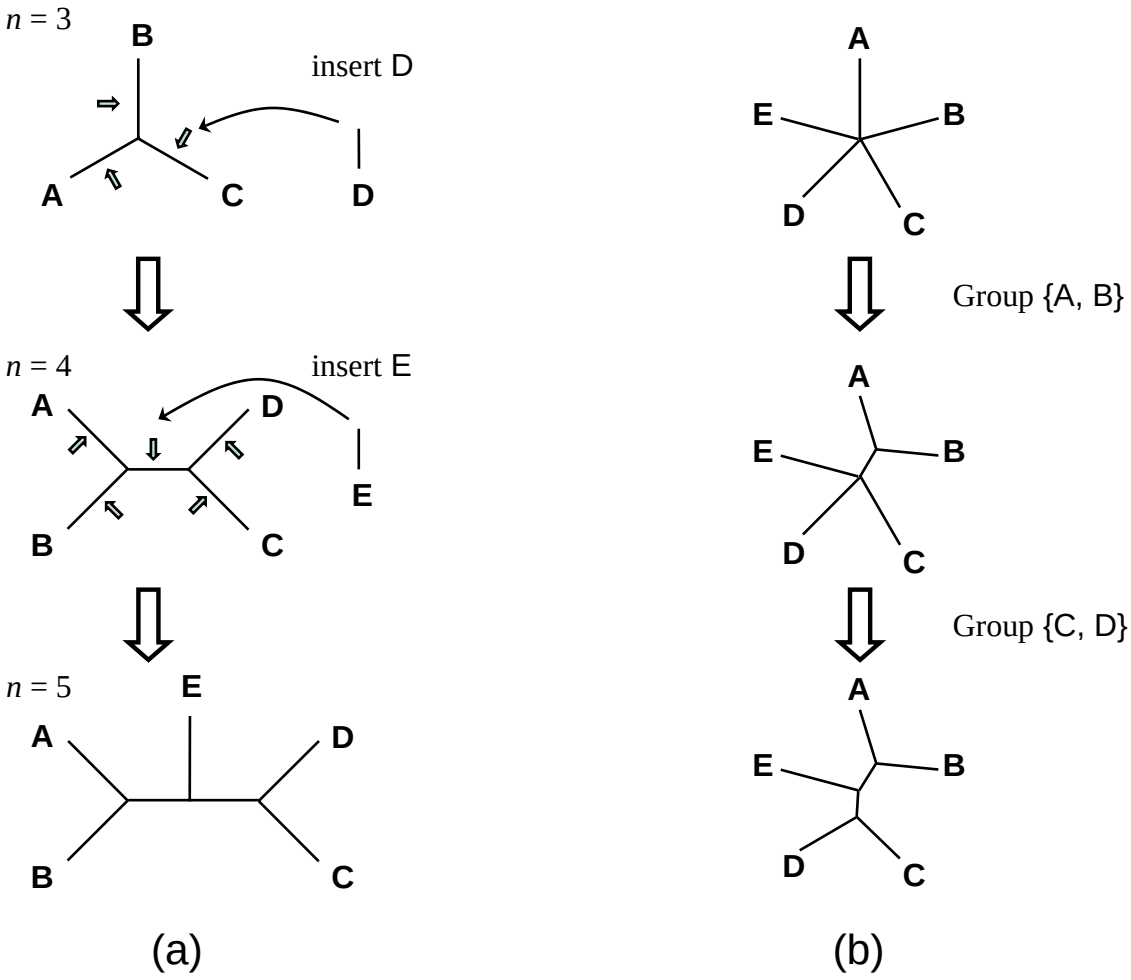


Figure 2.3: Generate initial trees. (a) Sequential Addition. (b) Star decomposition.

strate the basic concept of these techniques. There are other variations that modify these tree rearrangement techniques.

Subtree pruning and regrafting

The basic idea of subtree pruning and regrafting is removing a subtree from the original tree and reattaching this subtree back to somewhere else in the tree. As shown in Fig. 2.4, a subtree with 3 species is removed. The remaining subtree will reform a tree. The next step is illustrated in Fig. 2.5. The removed subtree can be attached to one of the

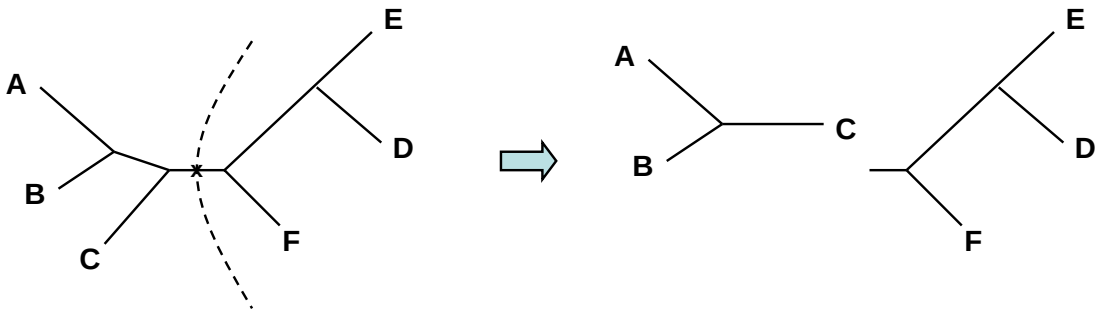


Figure 2.4: Pruning step: a subtree (righthand side) with 3 species is removed; the remaining subtree reforms a tree.

three branches of the remaining tree. Note that the branch connected to species C is the original location. Attach the subtree to the other branches forms a different tree topology from the original one.

Tree bisection and reconnection

Similar to SPR, the first step of tree bisection and reconnection is to break a branch and form two subtrees. However, these two subtrees will reform two trees as shown in Fig. 2.6. Reconnect these two trees by placing a branch between all possible branches in one tree and all possible branches in the other. As shown in Fig. 2.7, a new tree can be generated by selecting one of all possible combinations.

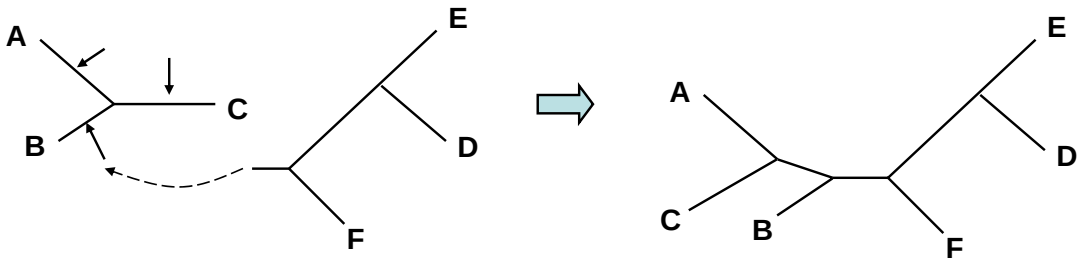


Figure 2.5: Regrafting step: the removed subtree is reinserted to the branch connected to species B.

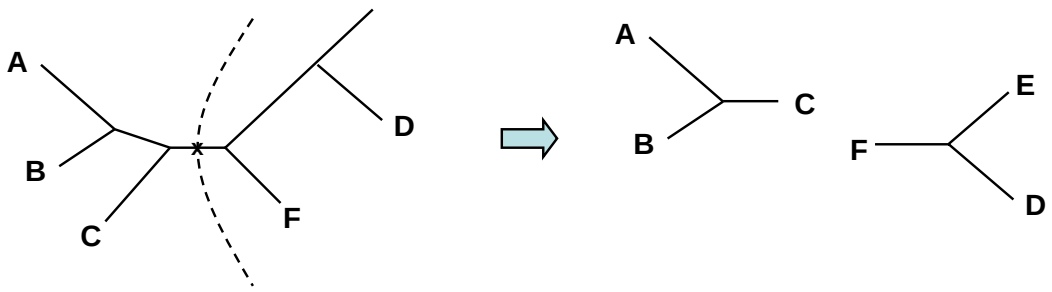


Figure 2.6: Bisection step: a branch is broken and two trees are reformed.

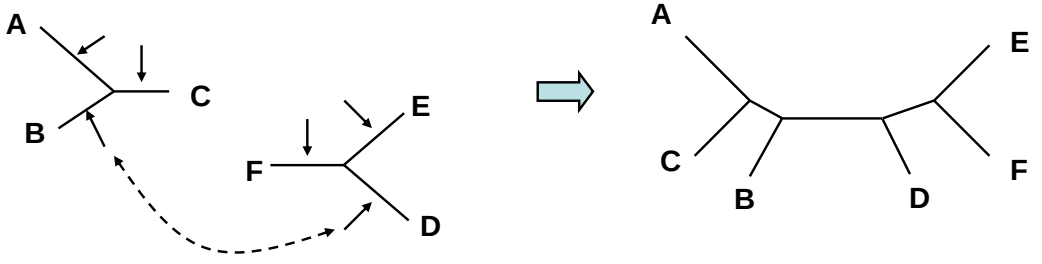


Figure 2.7: Reconnection step: reconnect these two trees by placing a branch between the branch to species B and the branch to species D.

Nearest-neighbor interchanges

The basic idea of nearest-neighbor interchanges is to swap two adjacent branches in a tree. As indicated in Fig. 2.8. Any tree with more than 3 species can be viewed as a 4-subtree quartet by randomly selecting an interior branch. Then, we can dissolve the

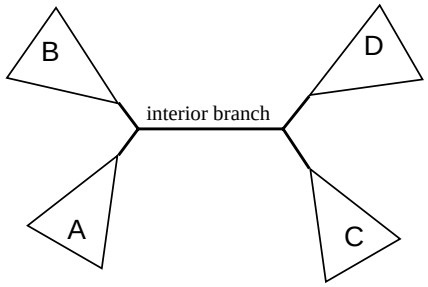


Figure 2.8: NNI: a 4-subtree quartet tree.

connections to the interior branch and remove the interior branch as shown in Fig. 2.9. The interior connections can be reformed in three different ways. One of them will form the original tree. The other two outcomes are shown in Fig. 2.10.

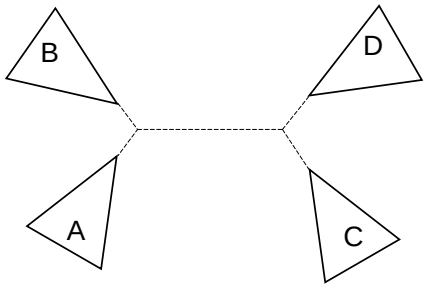


Figure 2.9: NNI: the interior connections are dissolved.

Tree-fusing

The idea of tree-fusing is quite simple. Assume we have two trees with different topologies and both of them have a subtree that contains the same list of species. For example, in Fig. 2.11, both trees have a subtree contains species A, B, C and D. The tree on top contains subtree ((A,B), (C,D)) and T_1. The tree at the bottom contains subtree ((A,C),

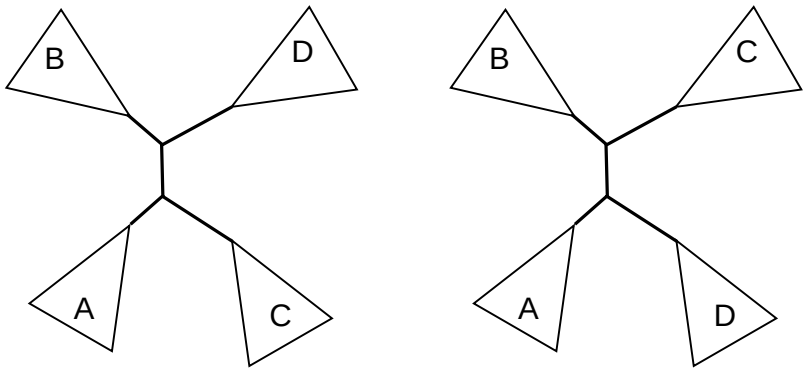


Figure 2.10: NNI: reform the interior connections in different ways.

(B,D)) and T_2. Two new trees are then generated by swapping the subtrees ((A,B), (C,D)) and ((A,C), (B,D)). This method sometimes is referred to as *subtree-swapping* method with variational implementations.

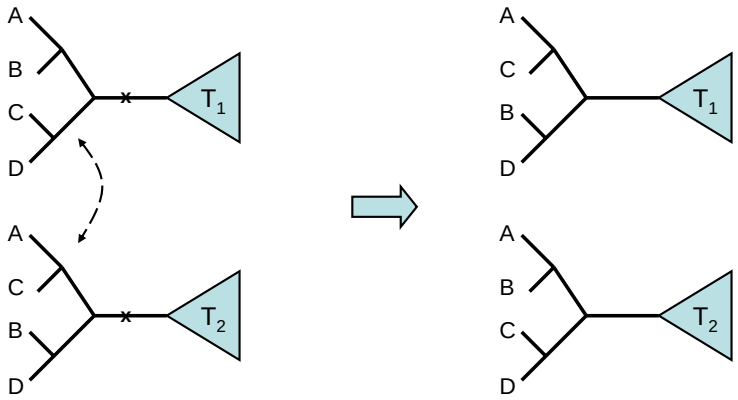


Figure 2.11: An example of tree-fusing.

2.4 Literature Review of Topology Searching Methods

When the size of trees becomes large, the search for the maximum-parsimony or maximum-likelihood trees gets more difficult. Hendy and Penny proposed a branch-and-bound algorithm, which utilized a sequential addition procedure with information of

bounds on each partial tree [30]. They have shown that their branch-and-bound algorithm can guarantee the global optimal maximum-parsimony trees. However, the algorithm is excessively slow when the size of phylogenetic trees becomes large. A way to improve the efficiency of branch-and-bound algorithm is to improve the quality of the bounds. Hendy and Penny described the importance of the order used in the sequential addition procedure. Purdom *et al.* demonstrated the improvements by re-evaluating the order in the sequential addition procedure during the branch-and-bound searching process [47].

Heuristic searches have been a focus for the past 20 years. *Hill-climbing* methods are widely used and they are usually combined with strategies that can generate good starting trees. *DNAPARS* and *DNAML* programs, included in the widely-distributed package *PHYLIP* created by Felsenstein [18], are both using the sequential addition technique with local tree rearrangements (NNI) to find most parsimonious and maximum-likelihood trees, respectively. Olsen *et al.* created *fastDNAML*, which is very similar to DNAML. It can outperform DNAML by using of a different optimization procedure on branch lengths [46]. It also utilizes SPR to perform the tree rearrangements. Yang [61] created the program, *PAML*, which employees both sequential addition and star decomposition methods to produce starting trees. Then NNI is performed as tree rearrangements to search for better maximum-likelihood trees. *PAUP** is another well-known program, which also uses sequential addition technique to find starting trees [57]. Both NNI and TBR are applied to the starting trees to perform tree rearrangements. This program can be used to find both of the maximum-parsimonious and maximum-likelihood trees. Guindon and Gascuel proposed their program, *PHYML*, in 2003, which might be one of the fastest program using hill-climbing methods [28]. They adopted a subtree swapping method, which enable them to efficiently evaluate new trees using the information from the current trees. Another fast program using hill-climbing method is *RAxML-III* created by Stamatakis *et al.* [55]. It uses the trees generated from DNAPARS as the starting trees. Then a rearrangement technique similar to fastDNAML is performed. Instead of optimizing all branch lengths of the new tree, RAxML-III only optimizes the branch lengths nearby the inserted branches. This strategy greatly improves the efficiency of this program.

A simulated annealing method was firstly proposed by Lundy to search for optimal parsimony trees [39]. Dress and Krüger also used simulated annealing with subtree-swapping as their tree rearrangement method [12]. Goloboff proposed a "tree-drifting" method on

finding the most parsimonious trees [26]. His method implements simulated annealing with a "*relative fit difference*" measure, which can emphasize small differences in parsimony score, to decide if a tree is accepted. He also applied the *sectorial search* to his TBR rearrangement and greatly improved search efficiency. Barker created a computer program, named LVB, using simulated annealing to search for the most parsimonious trees [1]. Salter and Pearl used the simulated annealing to search for maximum-likelihood trees [50]. Stamatakis created *RAxML-SA*, which combines both hill-climbing and simulated annealing methods, on finding maximum-likelihood trees [54]. Most part of this program is similar to RAxML-III except that a new tree is accepted or rejected based on a simulated annealing design.

Matsuda is probably the first to apply genetic algorithms on phylogeny study [40]. He used subtree-swapping to perform crossovers between trees. Lewis proposed his genetic algorithm program, *GAML*, which uses a revised SPR rearrangement as the crossover operator [37]. Moilanen used a similar crossover operator to Lewis's and used SPR as mutation operator to find maximum-parsimonious trees [44]. Katoh *et al.* used TBR as mutation operator and tree-fusing as crossover operator to search maximum-likelihood trees [34]. Congdon used a crossover operator similar to Lewis's and a leaf-node-swapping method as mutation operator on finding maximum-parsimonious trees [9]. Brauer *et al.* applied Lewis's algorithm to parallel computing and made it possible for their program to analyze phylogenetic trees with hundreds of sequences. Another program, *METAPIGA* created by Lemmon and Milinkovitch [36], performs genetic algorithms on multiple populations, called metapopulation, which can evolve in parallel. The populations near to each other are forced to "cooperate" in search of optimal trees. Zwickl introduced a genetic-algorithm program, *GARLI*, which is based on Lewis's GAML [63]. Both NNI and SPR are served as mutation operator in this program. In this SPR, the pruned subtree may only be re-inserted to the branches within a certain range from its original location.

In addition to the three main categories of heuristic search methods, we will briefly introduce some new methods developed in recent years. Charleston introduced his *Hitchhiking* strategy, which is a parallel search strategy with a random-walk search scheme [8]. The tree rearrangements taking on one tree will be propagated to a number of others in the same group. A *scatter search* method with *path relinking* was proposed by Cotta in 2004 [10]. It is a population-based search strategy, which uses a deterministic approach,

called path relinking, to recombine trees in order to generate new trees. Strimmer and von Haeseler proposed a new topology search strategy, called *quartet puzzling* in 1996 [56]. This algorithm consists of three steps. First, all possible quartet trees, i.e. 4-species trees, are constructed. Then the quartet trees are repeatedly combined together to form a complete tree. The final step is to summarize the majority rule consensus of all intermediate trees resulting from the combination step of quartet trees. In 2004, Vinh and von Haeseler created a program, *IQPNNI*, which extends the quartet puzzling algorithm [58]. The revised quartet puzzling algorithm is called *important quartet puzzling*, which performs the combination on part of quartet trees instead of all of them. The resulting trees are further improved by using NNI method. Minh *et al.* created *pIQPNNI* in 2005, which is a parallel-computation version of IQPNNI [43].

Studies have shown that reweighting or partition the characters in the sequence data usually improves the efficiency of topology searching. Holland *et al.* proposed a *MinMax squeeze* method, which uses partitions of the aligned sequences to construct better lower bounds for the branch-and-bound algorithm [31]. A *parsimony ratchet* method was proposed by Nixon, which reweights characters on a randomly selected subset of sequence data to propose new trees [45]. Then TBR and a *hill-climbing* search method are used to find maximum-parsimonious trees. Quicke *et al.* proposed a similar reweighting scheme on finding maximum-parsionmious trees [48]. Instead of reweighting characters randomly, they emphasized characters that fit the current best trees.

Chapter 3

Tabu Search for Phylogeny Inference

3.1 Introduction

Tabu search is a strategy for solving combinatorial optimization problems [25]. In the past two decades, tabu search has been successfully applied to various combinatorial problems including the traveling salesman problem [62]. However, its use for solving problems involving biological data is just being explored [3].

Fig. 3.1 demonstrates the procedure of a tabu search method. It searches the solution space by moving iteratively to the best neighbor of a current solution even if this leads to a deteriorated solution. In order to prevent getting trapped in locally optimal solutions, some tabu restrictions and aspiration criteria are used to constrain and guide the search process. By imposing tabu restrictions, some solutions that were recently visited are treated as inaccessible for a number of iterations while aspiration criteria allow exceptions for these restrictions. As a result, the tabu search proceeds by replacing the current solution at each iteration with the best accessible neighbor. The inaccessible solutions are stored in a *tabu list*. The *tabu tenure* defines the number of iterations the solutions stay on the tabu list. The inaccessible solutions become accessible after their tabu tenures have expired. Thus, tabu search can be viewed as a search strategy with an *adaptive* and *recency-based* memory which allows the searching process to be more efficient. The basic procedure for implementing tabu search is described in Algorithm 3.1.

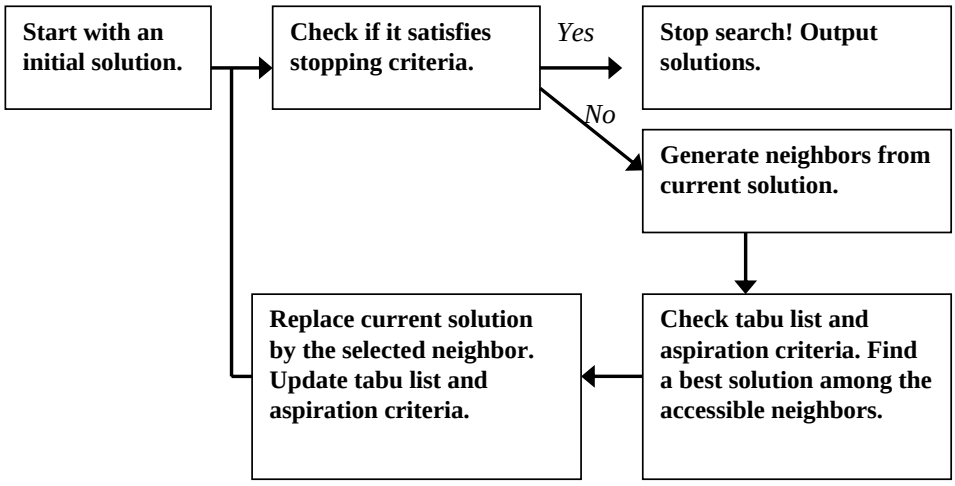


Figure 3.1: The process of tabu search.

Algorithm 3.1 Basic tabu search procedure

Initialize:
    Generate initial solution: x_0.
    Update best solution: $x^* = x_0$.
    Set iteration counter: $i = 0$.
    Set tabu list: *tabu list* $= \phi$.

Do while (stopping conditions not met)
    Generate new solutions: $V \subseteq Neighborhood(x_i)$.
    Choose best $v^* \in V$ and let *found_next_solution* = False.
    Do while (*found_next_solution* = False)
        If $v^* \in$ *tabu list* and $aspiration(v^*) \neq$ True
            Choose next best $v^* \in V$.
        End
        Else
            Let *found_next_solution* = True.
        End
    End
    Set $x_{i+1} = v^*$.
    If $Objective(x_{i+1})$ is better than $Objective(x^*)$
        Let $x^* = x_{i+1}$.
    End
    Update *tabu list*.
    Set $i = i + 1$.
End

To design our tabu search algorithm for phylogeny inference, two major issues should be addressed: (1) how to represent a tree; and (2) how to generate the neighbors of a current tree. In the following section, we propose a two-array structure to represent phylogenetic tree topologies. Then we introduced two methods to generate neighboring trees based on the two-array structure. The implementation of the tabu search algorithm for finding the maximum-parsimony phylogenetic trees are described in Section 3.4.

3.2 Representation of a tree

We have already introduced the basic structure of a phylogenetic tree topology in Chapter 2. Given a rooted, binary tree with n leaves, we label these leaves by $1, 2, \ldots, n$. Motivated by Mau, Newton and Larget [41], we define an array $L = \{L_1, L_2, \ldots, L_n\}$ to represent the horizontal position of each leaf, i.e., L_1 is the left-most leaf while L_n is the right-most one. Since there are $n-1$ interior nodes in an n-leaf rooted tree, we label them by $n+1, n+2, \ldots, 2n-1$, which indicates the horizontal position of each interior node as well, i.e., $n+1$ is the left-most interior node while $2n-1$ is the right-most interior node. Fig. 3.2 shows an example of this representation. We can observe that there is always an interior node between two leaves. For example, node 6 is between leaves 2 and 3, and node 8 is between leaves 1 and 5. Moreover, we define an array $N = \{N_1, N_2, \ldots, N_{n-1}\}$ to represent the vertical position of each interior node, i.e. N_1 is the root of the tree and N_{n-1} is the lowest interior node. Since we focus on maximum-parsimony phylogeny inference first, we do not consider the branch lengths of each tree at this moment.

Note that there is a parental node for each interior node and leaf except the root and there are two descendants (referred to as the left child and the right child) for each interior node. We can use the structure as shown in Fig. 3.3 to store the information for each interior node and leaf. It is clear to see that for any given arrays of $L = \{L_1, L_2, \ldots, L_n\}$ and $N = \{N_1, N_2, \ldots, N_{n-1}\}$, there is a unique rooted, binary tree corresponding to it. In order to reconstruct a tree from given L and N arrays, we need to fill in all required information in the structure of Fig. 3.3. for each interior node and leaf. A traversal mechanism demonstrated below helps us to accomplish this task (as illustrated in Fig. 3.4). Start with the root node (the highest node, node 7) and draw the branches to the highest node to the left (node 6) and the right (node 8) of the root. After two descending branches

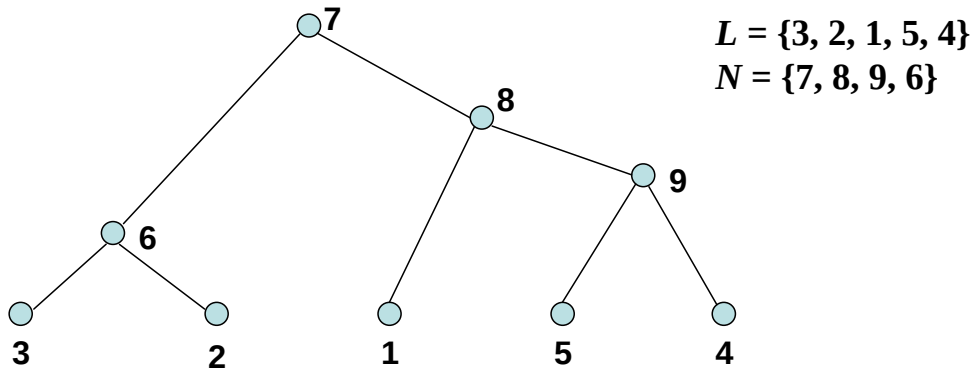


Figure 3.2: The representation for a rooted, binary tree.

are drawn, traverse downward on the tree onto one of the descending nodes of current node. Now, suppose we have reached node k, say node 8. The child nodes of k must lie within the horizontal positions bounded by any node that is higher than k. In this case, it is node 7 and the boundary is drawn as the dashed line in Fig. 3.4. As before, draw the branches to the highest nodes to the left and right within this boundary (1 and 9). Notice that there is no interior node on the left hand side of node 8 within the boundary. Since we know the horizontal positions of all the nodes from array L and the labels of interior nodes, we found that the left child of node 8 should be leaf 1. This traversal procedure continues until all the leaves are reached.

3.3 Generate neighbors

After knowing the representation of a tree, we are ready to introduce two mechanisms for generating the neighboring trees from a current tree. One is called *profile change* and the other *leaf swapping*. The former involves swapping two elements in array N and the latter, L. By swapping two elements in array N, we can shift the vertical position of interior nodes up and down (as shown in Fig. 3.5(a)). Similarly, we can change the horizontal positions of two leaves by swapping two elements in array L (as shown in Fig. 3.5(b)). Then, the traversal procedure described in the previous section helps us to construct a neighboring

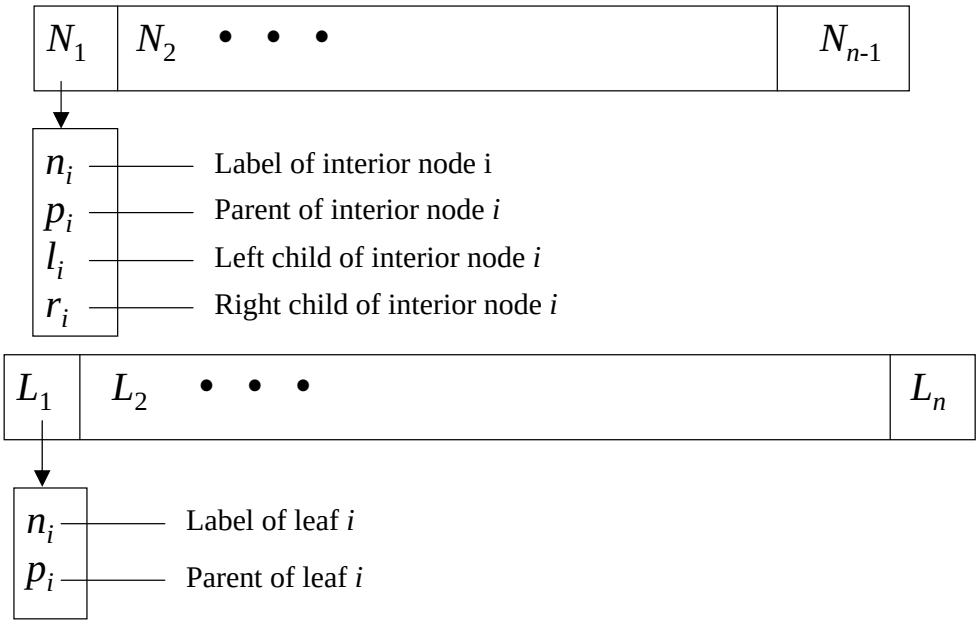


Figure 3.3: The structure to store information for each leaf and interior node.

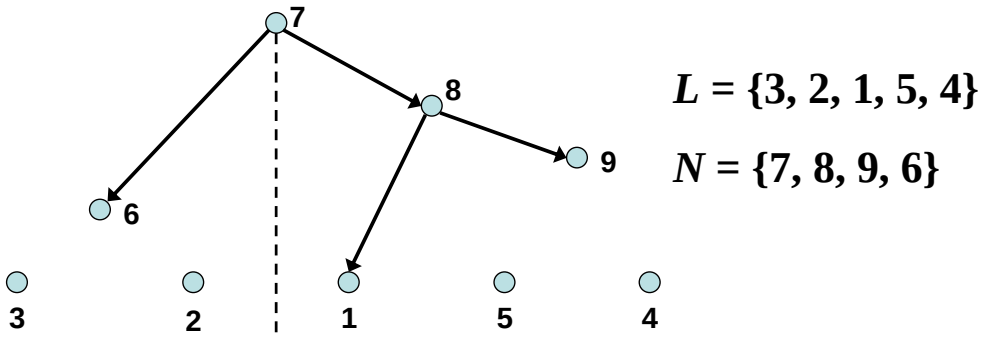


Figure 3.4: The traversal procedure to reconstruct the tree.

tree with different topology.

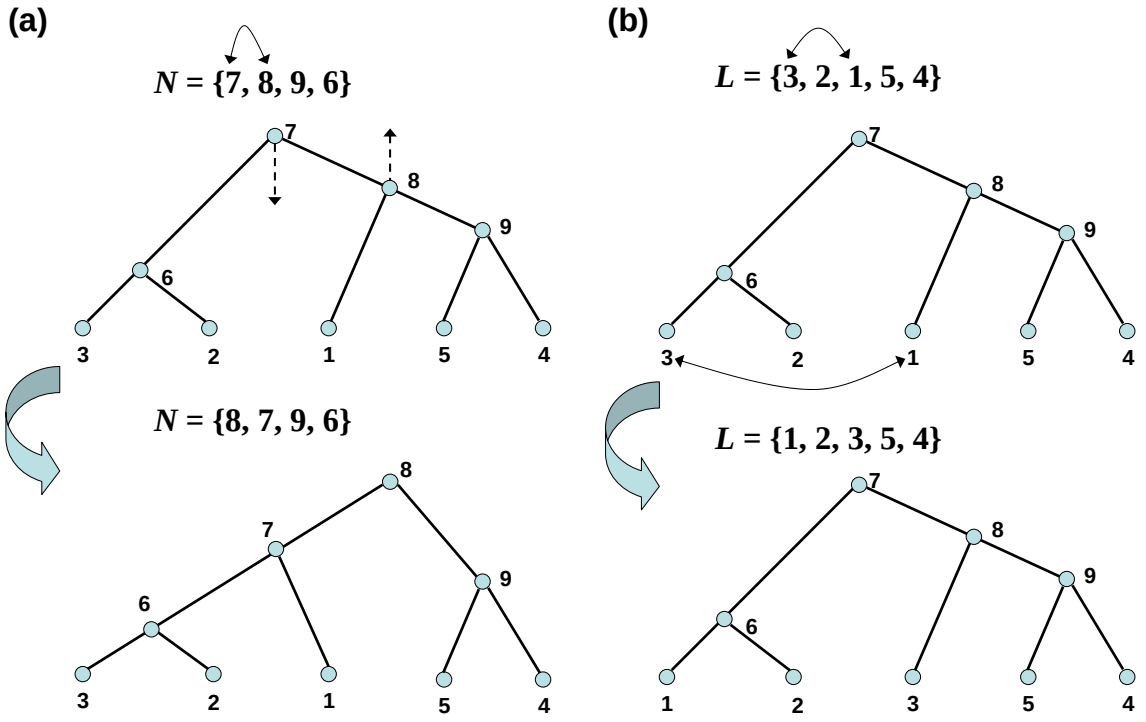


Figure 3.5: (a) Profile change. (b) Leaf swapping.

Notice that the two mechanisms may produce a neighbor with the same topology as the current tree. For example, if we swap nodes 6 and 9 in N or swap leaves 2 and 3 in L as shown in Fig. 3.6, the topology will remain the same. Therefore, we need some restrictions when applying these two operations. For the profile change, first, we select a node, say node k, in N for swapping. Then pick another node in N that is higher than the parent of k (or the parent itself), or lower than the child nodes of k (or one of the child node itself). For the leaf swapping, we check the selected leaves to see if they have the same parent. If so, then select another pair of leaves for swapping. By imposing these restrictions, we can ensure that the neighboring trees will have different topologies from the current tree.

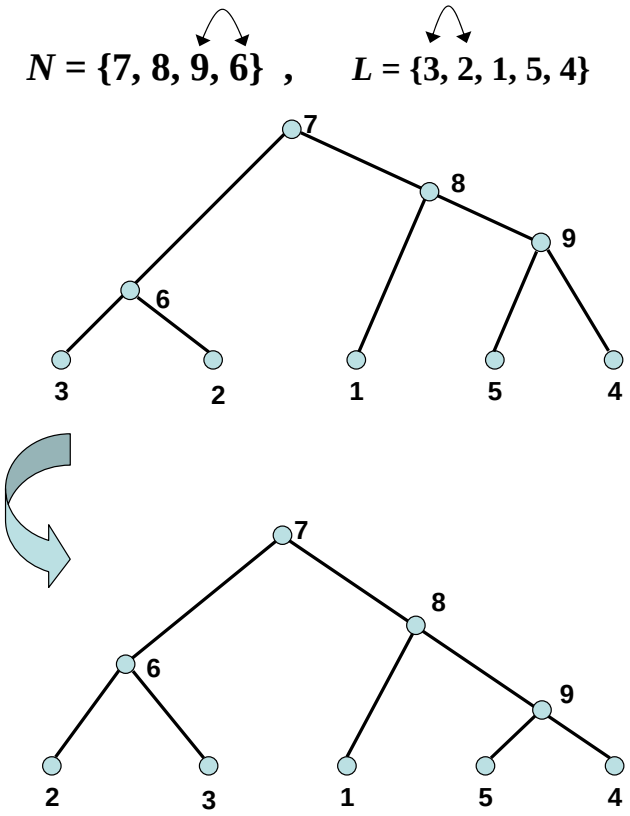


Figure 3.6: The topology remains the same after profile change and leaf swapping.

3.4 Tabu search algorithm for finding maximum-parsimony trees

As we described before, a tabu search algorithm requires tabu restrictions and aspiration criteria. In order to prevent the searching process from revisiting a tree that has been visited recently, we keep a tabu list to store the information of recent steps. Specifically, we store the L and N arrays of recently visited trees for a number of iterations. A tree with exactly the same L and N arrays as some tree stored in the tabu list is considered inaccessible until its tabu tenure has expired and it is released from the tabu list. Notice that two trees with the same topology may be described in different L and N arrays. Accordingly, a tree with the same topology as another tree in the tabu list may still be accessible. However, this does not bother us because the trees with different L and N arrays will generate different neighbors even when they have the same topology. In fact, this allows us to have more flexibility in the searching process without greatly increasing the chance of being trapped in a cycle. The minimum evolutionary changes of each tree will be evaluated based on Fitch's algorithm and the value will be referred to as parsimony score. The aspiration criteria are chosen according to the parsimony score of a tree. In other words, if a tree, say i, satisfies the following condition: $S_i \leq S^* + \varepsilon$, where S_i denotes the parsimony score of tree i, S^* is the best parsimony score we have so far, and ε is some nonnegative value (may be set as zero), then tree i will be allowed to be revisited, whether it is on the tabu list or not. Therefore, we can define the set of "accessible trees" to be the union of those that are not in the tabu list and those that are in the tabu list but satisfy the aspiration criteria.

In the very beginning of our tabu search algorithm, we generate an initial tree with a random topology (e.g. $L = \{1, 2, \ldots, n\}$ and $N = \{n+1, n+2, \ldots, 2n-1\}$), and calculate its parsimony score. In each of later iterations, we randomly generate a number of neighbors and select the one with the smallest parsimony score among those that are "accessible". Then, we replace the current tree by the selected tree and update the tabu list. In addition to the tabu list, we also keep a best-solution list to store the best trees we have found so far during the searching process.

In our design, we use three different ways to generate neighbors. First, start with

leaf swapping to generate n neighbors. If we cannot find an improved solution (i.e., a tree with lower parsimony score than the best parsimony score we found so far) after k iterations, then we activate profile change to generate n neighbors. Similarly, if the solutions are not improved after k iterations, we switch back to use leaf swapping. We repeat these two operations until an improved solution occurs. However, if the best parsimony score is not improved after a long period of time, say, k' iterations, then we apply both the leaf swapping and profile change to generate more neighbors, say, n', in order to escape from being trapped in the local optima. We switch back to use the leaf swapping after k iterations without improving the parsimony scores. The above procedure is defined as a search cycle. A search cycle will be reset if an improved solution is found, or it will be repeated until we terminate the program. If the solution cannot be improved in a very long period of time, then the program will be terminated and output the best trees we found. The procedure of the tabu search for maximum-parsimony phylogeny inference is shown in Algorithm 3.2.

Algorithm 3.2 Tabu search for maximum-parsimony phylogeny inference

Initialize:
Randomly generate an initial tree : t_0.
Update the best tree list: $T^* = t_0$.
Update the best parsimony score: $S^* = S_{t_0}$.
Set iteration counter: $i = 0$.
Set tabu list: *tabu list* $= \phi$.
Activate *leaf swapping*.

Do while (stopping conditions not met)
Generate new solutions, $\tilde{T}$, with activated tree rearrangement methods.
Choose the best $\tilde{t}^* \in \tilde{T}$ and let *found_next_solution* = False.

Do while (*found_next_solution* = False)
If $\tilde{t}^* \in$ *tabu list* and $S_{\tilde{t}^*} > S^* + \varepsilon$
Choose the next best $\tilde{t}^* \in \tilde{T}$.
End
Else
Let *found_next_solution* = True.
End
End

Set $t_{i+1} = \tilde{t}^*$.

If $S_{t_{i+1}} < S^*$
Let $T^* = t_{i+1}$.
End
Else if $S_{t_{i+1}} = S^*$
Let $T^* = T^* \cup t_{i+1}$.
End

Update *tabu list*.
Set $i = i + 1$.

Algorithm 3.2 (Cont'd)

If both *profile change* and *leaf swapping* activated
  If no improvement for more than k iterations
    Deactivate *profile change* and keep *leaf swapping* activated.
    Set the number of new generated solutions = n.
  End
End
Else if only *leaf swapping* is activated
  If no improvement for more than k' iterations
    Activate *profile change*.
    Set the number of new generated solutions = n'.
  End
  Else if no improvement for more than k iterations
    Deactivate *leaf swapping* and activate *profile change*.
  End
End
Else
  If no improvement for more than k' iterations
    Activate *leaf swapping*.
    Set the number of new generated solutions = n'.
  End
  Else if no improvement for more than k iterations
    Deactivate *profile change* and activate *leaf swapping*.
  End
End
End

Chapter 4

Performance of the Tabu Search Algorithm

4.1 Introduction

In order to know if our tabu search algorithm can find the maximum-parsimony phylogenetic trees, we use the results generated from the program *dnapenny* as benchmark for comparisons. The dnapenny program was created by Felsenstein in his PHYLIP package [18], and it implements a branch-and-bound algorithm for finding the most parsimonious trees. It can guarantee global optimal maximum-parsimony trees. This program also employs Fitch's algorithm to calculate the parsimony score and therefore the results should be comparable with those from the tabu search program.

The data sets we use are randomly sampled from a 47-taxon DNA sequence data set with 1017 nucleic acid characters [24]. We randomly drew m ($10 \leq m \leq 20$) sequences from the whole 47 sequences to generate one data set, and 10 data sets were randomly generated for each m. Our comparisons are based on the average performance.

In order to implement the tabu search algorithm on phylogeny reconstruction problems, we create a program *TABUPARS* (tabu search for parsimony phylogeny) in the C++ language. The followings are the detailed settings of the parameters as described in section 3.4: n = the number of taxa involved, $k = 10$, $k' = 150$ and $n' = \frac{n^3}{3}$. These values were determined after conducting some tuning tests.

4.2 Experimental Results

We use a different philosophy to compare the results instead of using CPU time. First, we run the dnapenny program to identify a global optimal solution. Then we run both dnapenny and tabupars programs but terminate them immediately after reaching the optimal parsimony score. We count the number of trees these two programs evaluated before reaching the optimal solution. The results are grouped based on the number of taxa. The average of each group for number of taxa ranging from 10 to 20 is shown in Table 4.1. Fig. 4.1 shows the trend of the difference of results between these two programs.

The result shown in Fig. 4.1 indicates that as the number of taxa increases, the effectiveness of the tabupars program dominates that of the dnapenny program. Another important result is when the number of taxa goes beyond 20, the dnapenny program cannot always guarantee global optima with its maximal run-length (evaluates 900,000,000 trees). For example, when the number of taxa is 21, 2 out of 10 data sets only come out with a local optimal solution. The tabupars program actually achieved better solutions in these two cases with less than 1,000,000 trees evaluated.

Table 4.1: The average number of trees evaluated before reaching the optimal solution.

number of taxa	tabupars	dnapenny	ratio*
10	4,878	1,396	3.4943
11	3,720	12,755	0.2917
12	4,574	7,865	0.5822
13	14,059	30,927	0.4546
14	9,131	115,607	0.0789
15	40,914	81,028	0.5049
16	196,995	84,559	2.3297
17	84,484	277,417	0.3045
18	156,351	9,482,194	0.0165
19	453,356	15,060,596	0.0301
20	321,753	79,295,570	0.0041

* ratio = $\frac{\text{tabupars}}{\text{dnapenny}}$.

The percentages of solution space searched by the tabupars program is defined as

$$100 \times \frac{\text{number of parsimony evaluations}}{\text{number of possible tree topologies}}. \tag{4.1}$$

Table 4.2 demonstrates the percentage of solution space searched by the tabupars program

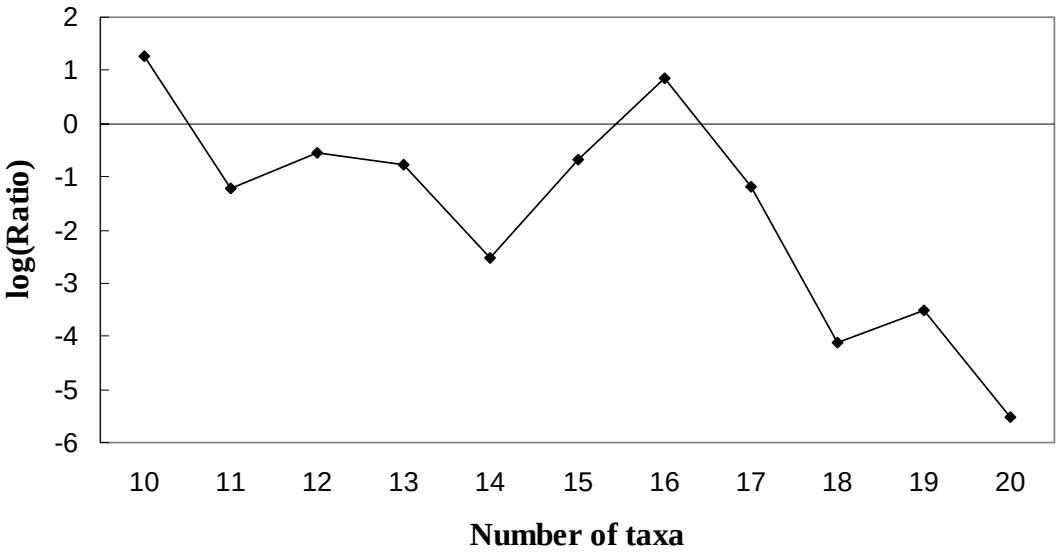


Figure 4.1: The trend of log(ratio) of the results between these two programs.

before reaching the most parsimonious trees. We could see that the value of the percentage decreases drastically as the number of taxa increases. The trend in Fig. 4.2 shown an exponential decreasing rate as the number of taxa increases.

4.3 Discussion

Our experiments have shown that the tabupars program requires the evaluation of fewer topologies to find a most parsimonious tree than the dnapenny program. Although our comparison scheme is limited to small data sets, Table 4.2 provides a hint to stop the tabu search in a pre-specified number of iterations to reach a good approximate solution. One issue arises in the comparisons between the tabupars and dnapenny programs is that the global optimal parsimony scores of the testing data sets are known. The comparison scheme allows us to assess the capability and efficiency of the tabupars program finding the global optima. However, in a real search for the maximum-parsimony phylogenetic trees for a median-sized or large-sized data set, the global optimum is usually unknown. As a result, the search continues after the globally optimum topology is evaluated. Therefore, the comparison scheme we use does not reflect strengths or weaknesses of the terminating

Table 4.2: The average percentage of solution space searched by the tabupars program before reaching the optimal maximum-parsimony trees.

number of taxa	% of trees
10	1.41558E-02
11	5.68189E-04
12	3.32686E-05
13	4.44582E-06
14	1.15507E-07
15	1.91671E-08
16	3.18232E-09
17	4.40254E-11
18	2.46896E-12
19	2.04543E-13
20	3.92344E-15

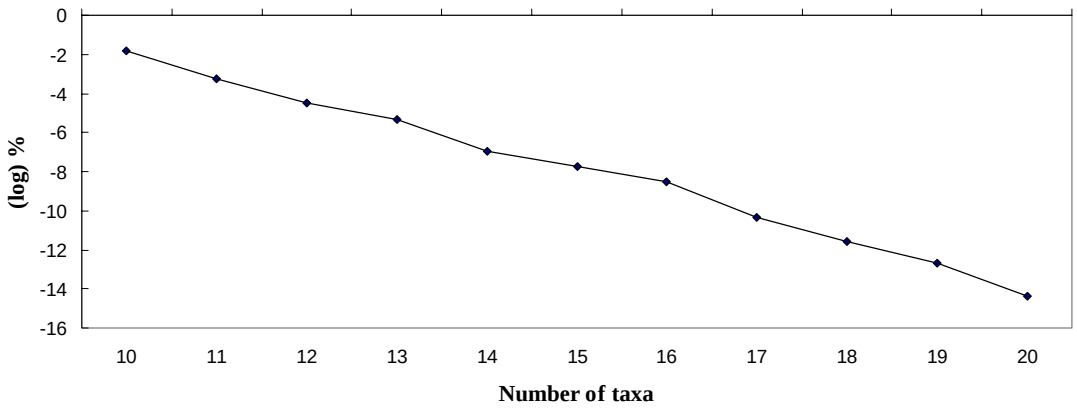


Figure 4.2: The trend of the log-percentage of solution space searched by the tabupars program before reaching the most parsimonious tree.

criteria of topology search methods. In Chapter 6, we present a different comparison scheme which allows us to compare different phylogenetic programs without knowing the global optimal parsimony scores beforehand. Another issue we need to be concerned about is the efficiency of the tabupars program. The experimental results for the small data sets are acceptable. However, the tabupars program became too slow to converge when the data size is large. There are several possible reasons of the slow convergency. First, the tree representation is not a one-to-one representation. The redundance may decrease the effectiveness of the tabu search method. Second, when the solution space becomes large, the frequency of a tree been tabooed decreases. This may also decrease the effectiveness of the tabu search method. In consequence, we proposed a new approach in the following chapter. We will also investigate the improvement of this new approach.

Chapter 5

Hybrid Method of Genetic Algorithm and Tabu Search for Maximum-parsimony Phylogeny Inference

5.1 Introduction

The development of genetic algorithms can be traced back to 1954 when Barricelli used a computer to simulate evolution [2]. Fraser extended the work on simulation of artificial selection, which includes most of the essential elements in genetic algorithms [22]. The first genetic algorithm applied to optimization problems was proposed by Bremermann in 1962 [4]. However, genetic algorithms did not draw much attention until Holland introduced a formalized framework to discuss the quality of solutions created in each generation [32].

Genetic algorithms are stochastic search algorithms that mimic the nature of evolution. Unlike conventional search techniques, genetic algorithms start with a set of solutions, called a *population*. Each individual in the population is called a *chromosome*, which represents a solution to the original problem. In each iteration, new chromosomes are generated either by recombining two chromosomes, called *crossover* or by modifying one chromosome, called *mutation*. Crossover and mutation are usually referred to as *genetic operators* and the

new chromosomes are called *offspring*. Crossover and mutation occur with probabilities, p_c and p_m, respectively, during each iteration of the search process. A new generation is then created by applying one or several selecting rules to choose chromosomes from offspring. Usually, a *fitter* chromosome in the offspring is more likely to be selected into the new generation. The fitness of each chromosome is evaluated based on the objective function of the original problem. Introduction to several commonly-used genetic operators, chromosome encoding methods, and selection techniques can be found in [23]. In Algorithm 5.1, we show the basic procedure for implementing a genetic algorithm.

Algorithm 5.1 Basic genetic algorithm procedure

Initialize:
    Generate initial population: P_0.
    Evaluate the initial population and find the best individual x_0^*.
    Update best solution: $x^* = x_0$.
    Set iteration counter: $i = 0$.

Do while (stopping conditions not met)
    Perform crossover operation on selected individuals of P_i: $C_i = Cr(P_i)$.
    Perform mutation operation on selected individuals of P_i: $M_i = Mu(P_i)$.
    Evaluate fitness of C_i and M_i and find the best individual $x_i^* \in C_i \cup M_i$.
    If $Objective(x_i^*)$ is better than $Objective(x^*)$
        Let $x^* = x_i^*$.
    End
    Select new population: $P_{i+1} = Select(P_i \cup C_i \cup M_i)$.
    Set $i = i + 1$.
End

In this chapter, we proposed a hybrid search method which combines genetic algorithm and tabu search for the maximum-parsimony phylogeny inference. Our goal is to improve the efficiency of the search process. In the following section, we described a different tree representation, which can avoid the redundancy caused by the previous tree representation. Our crossover and mutation operations are introduced in Section 5.3. The details of the implementation of our hybrid search method for finding maximum-parsimony is given in Section 5.4.

5.2 Tree Representation

In this section, we introduced a new tree representation which employees the canonical form of phylogenetic trees to create a one-to-one correspondence. The transformation of the canonical form includes two actions: re-rooting and branch rotation. They are described as follows. Assume we have a 4-leaf rooted tree as shown in Fig. 5.1 (a). Each leaf is labelled according to a pre-defined order. The position of the root does not matter if our objective is to find the maximum-parsimony trees. Therefore, we place the root on the branch which connects to the leaf with the smallest label. The tree after re-rooting is shown in Fig. 5.1 (b). The *Newick* format shown below the trees is a commonly-used tree representation, which consists of a series of nested parentheses, enclosing names and separated by commas [7]. We can see that the Newick format has been changed after re-rooting.

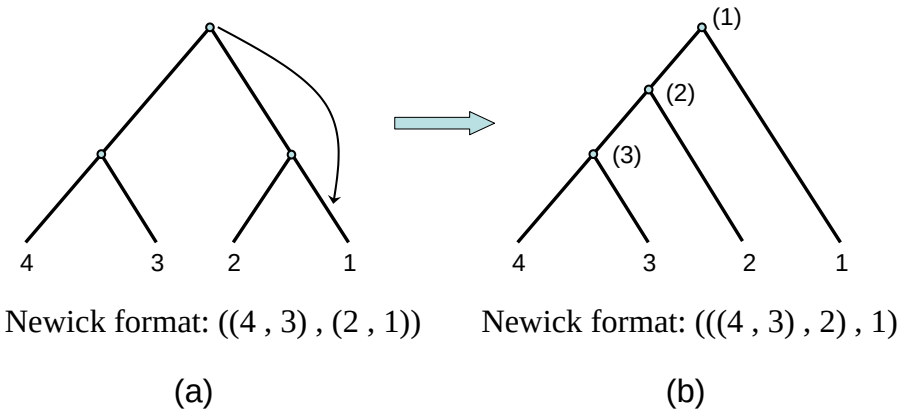


Figure 5.1: Re-root a 4-leaf tree.

We then label each interior node as the smallest-label leaf descending from it (see Fig. 5.1 (b)). The rotation of the branches does not affect the parsimony scores. Hence, for each interior node, we rotate the branches so that the descendant node with smaller label is on the left-hand branch. The tree shown in Fig. 5.2 (b) is the resulting tree of the tree in Fig. 5.2 (a) after branch rotation. This tree is also in the canonical form of all the trees shown in Fig. 5.1 and Fig. 5.2. The Newick format of the tree in canonical form is used as a *tree ID* to represent this tree. The advantages of this tree representation are two-fold. It

is a one-to-one correspondence and each tree is considered to be unrooted. The redundance can be avoided while the solution space is decreased (from rooted trees to unrooted trees).

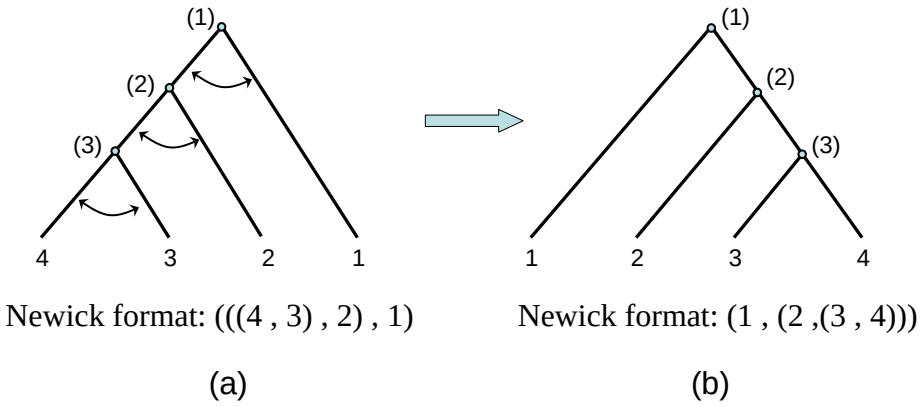


Figure 5.2: Rotate the branches of the tree in Fig. 5.2 (b).

5.3 Generating Neighbors

There are two operations, crossover and mutation, to generate offspring in our genetic algorithm. We employ *nearest neighbor interchange* (NNI) as the mutation operator and *subtree pruning and regrafting* (SPR) as the crossover operator. The SPR is modified to allow us to recombine two trees for new trees. The modifications of SPR are described as below. Note that each modified SPR performing on a pair of trees creates a pair of new trees .

Step 1. Pick two trees, tree A and tree B, as shown in 5.3 to perform a crossover operation. Randomly select a subtree, A2, on tree A and prune it off tree A.

Step 2. Remove the redundant branches on tree B in order to insert subtree A2 onto tree B. The mended tree is tree $B^{'}$ as shown in Fig. 5.4.

Step 3. Randomly select a branch of tree $B^{'}$ and attach subtree A2 to this branch. A new tree is then created as shown in Fig. 5.5.

Step 4. Apply the above steps on tree A and tree B again, but this time prune a randomly selected subtree from tree B and attach it on the mended tree A.

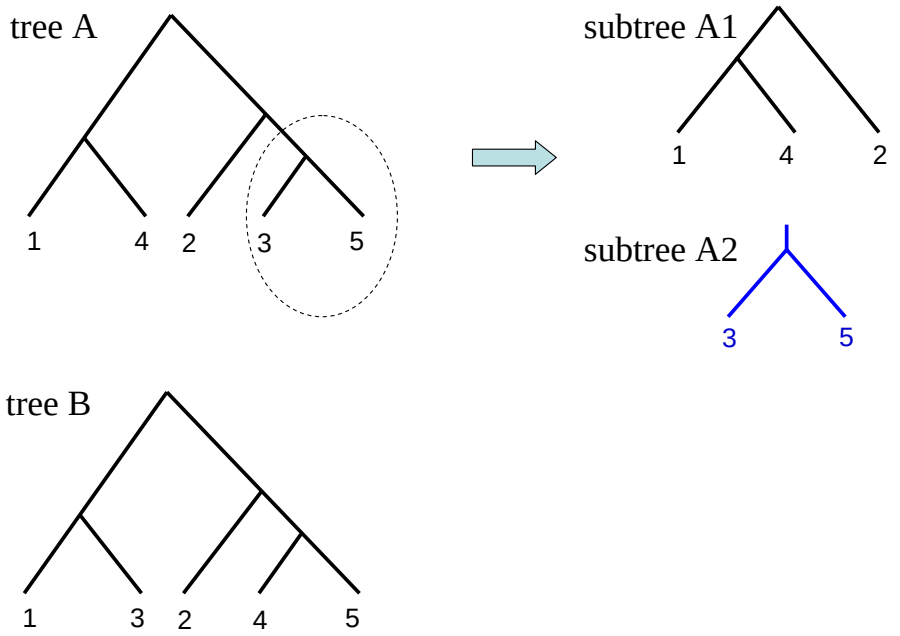


Figure 5.3: Modified SPR: pruning step.

The NNI tree rearrangement method described in Section 1.3.1 is adopted as the mutation operator. There are two types of mutation operations in our algorithm. Assume that a tree is selected for the mutation operation, then NNI tree rearrangement is performed on either one randomly selected interior branch or every interior branch of the selected tree. The former is called *one-branch-mutation*, and the later *all-branches-mutation*. The one-branch-mutation operator is the major mutation operator. The all-branches-mutation operator is used when an extensive search is performed on the best solutions we have found in order to escape from a local optimum.

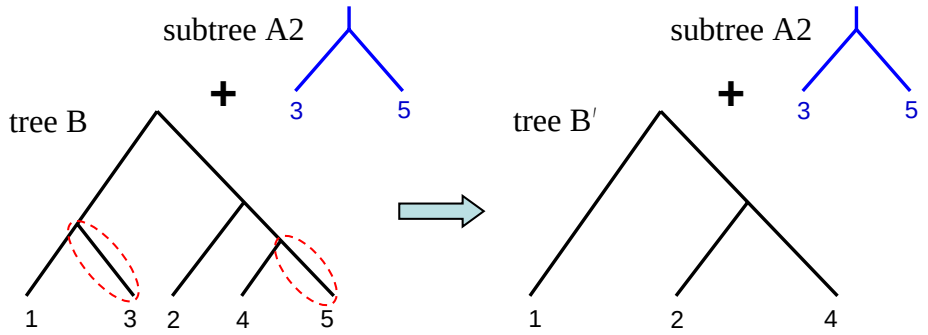


Figure 5.4: Modified SPR: mending step.

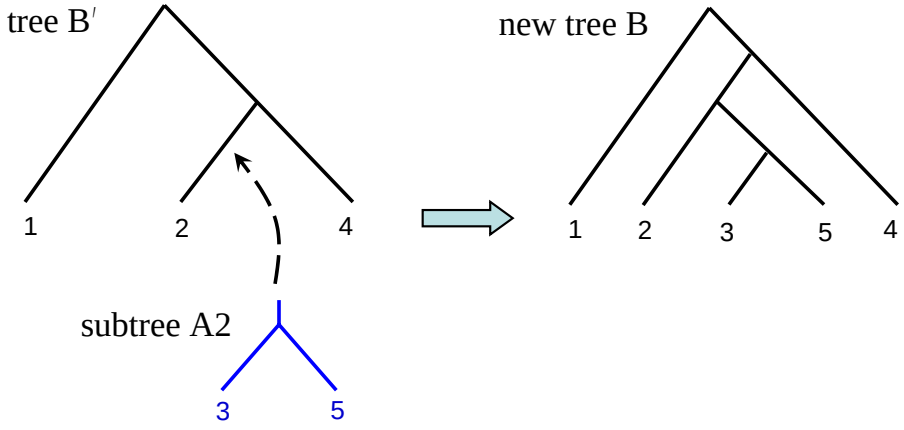


Figure 5.5: Modified SPR: grafting step.

5.4 Hybrid Method of Genetic Algorithm and Tabu Search for Finding Maximum-parsimony Trees

The distinct part of our genetic algorithm is incorporating the tabu search into selecting the next generations and creating the offspring. In the design of our genetic algorithm, the population will be divided into two groups: an *elite group* and a *non-elite group*. When creating the offspring, the crossover operator is performed on every pair of the members in the elite group. The one-branch-mutation operator is also performed on each member in the elite group to create some offspring. The rest of the offspring are created in two ways. Crossovers are carried out on randomly selected pairs of members in the non-elite group. And every member in the non-elite group will have a chance to have a one-branch-mutation operation with a pre-determined probability, called *mutation rate*.

After all the offspring have been created, they are ranked based on fitness. The fitness of each tree is evaluated by Fitch's parsimony algorithm. The selection for the elite group members begins from the top-ranked offspring. A tree which has been selected in recent iterations will not be selected into the elite group until its tabu tenure is expired. The selection of elite group members continues until the elite group is full. These new elite group members are then put on the tabu list and given a tabu tenure. The rest of the offspring are selected into the non-elite group according to a probability function of fitness value. Assume tree i has a parsimony score f_i and the best parsimony score found so far is f^*. The probability function is defined as

$$p(f_i) = 1 - \frac{f_i - f^* + \epsilon}{f_i}, \tag{5.1}$$

where ϵ is a small positive integer. It can be seen that the sum of the probability of the rest of the offspring is likely to be greater than 1. Therefore, the selection for the non-elite group members begins from the higher-ranked offspring and stops as soon as the non-elite group is full.

During the search process, the trees with the best parsimony score found so far are stored and recorded. If there is no improvement for a period of time, say T_{no}, we expand the offspring size by carrying out the all-branches-mutation on a portion of the newly found best trees. If there are no newly found best trees and there are more than two best trees recorded, crossovers will be performed on randomly selected pairs of the best trees. The

expansion of offspring size will not occur until another T_{no} period of non-improvement. If there is no improvement for a long period of time, say T_{stop}, we terminate the searching process and report the best trees we found. The procedure of our hybrid method for finding the maximum-parsimony trees is described in Algorithm 5.2.

Algorithm 5.2 Procedure for the hybrid method of GA and TS for maximum-parsimony phylogeny inference.

Initialize:
Generate an initial population: P_0, where $\|P_0\| = n_p$.
Evaluate the parsimony scores of the initial population.
Rank all trees in the population based on their parsimony scores and find the best trees t_0^*.
Update the best trees: $t^* = t_0^*$.
Update the best parsimony score: $f^* = f(t_0^*)$, where$f$ is the function for parsimony evaluation.
Select the top n_e trees into the elite group, $\mathscr{E}_0$.
Put the rest of the population in the non-elite group, $\mathscr{N}_0$.
Set *tabu list* $= \mathscr{E}_0$.
Set iteration counter: $i = 0$.

Do while (stopping conditions not met)
Set offspring: $C_i = \phi, M_i = \phi$.
Crossover on elite group: $\forall x, y \in \mathscr{E}_i$ and $x \neq y$, $C_i = C_i \cup Cr(x, y)$.
Mutation on elite group: $\forall x \in \mathscr{N}_i, M_i = M_i \cup Mu^1(x)$.

Do while (number of offspring $< n_o$)
Crossover on non-elite group: $Rand(x, y) \in \mathscr{N}_i$ and $x \neq y, C_i = C_i \cup Cr(x, y)$.
Mutation on non-elite group: $\forall x \in \mathscr{N}_i$, if $RN < p_m, M_i = M_i \cup Mu^1(x)$, where RN is a random number generated from random number generator.
End

If no improvement for T_{no} iterations
All-branches-mutation on best trees: $\forall x \in t^*, M_i = M_i \cup Mu^2(x)$.
End

Evaluate the parsimony scores of C_i and M_i.
Create the ranking for the offspring: $R = Rank(C_i \cup M_i)$.
Find the best trees $t_i^* \in C_i \cup M_i$.

If $f(t_i^*) < f^*$
$f^* = f(t_i^*)$ and $t^* = t_i^*$.
Else if $f(t_i^*) = f^*$
$t^* = t^* \cup t_i^*$.
End

Algorithm 5.2 Cont'd

Set new elite group and non-elite group: $\mathscr{E}_{i+1} = \phi, \mathscr{N}_{i+1} = \phi$.
Set $k = 1$.

Do while $(\|\mathscr{E}_{i+1}\| < n_e)$
  If $R(k) \notin$ *tabu list*
    $\mathscr{E}_{i+1} = \mathscr{E}_{i+1} \cup R(k)$.
    $R = R - \{R(k)\}$.
  End
  $k = k + 1$.
End

Reset $k = 1$.

Do while $\big(\|\mathscr{N}_{i+1}\| < (n_p - n_e)\big)$
  If $RN < 1 - \frac{f\big(R(k)\big) - f^* + \epsilon}{f\big(R(k)\big)}$
    $\mathscr{N}_{i+1} = \mathscr{N}_{i+1} \cup R(k)$.
  End
  $k = k + 1$.
End

Update *tabu list*.
Set $i = i + 1$.
End

Chapter 6

Performance of the Hybrid Method for Maximum-parsimony Phylogeny Inference

6.1 Introduction

In this chapter, we analyze the performance of the proposed hybrid method for finding maximum-parsimony trees. Two C++ programs, *GATSpars* and *GATSpars**, are created to implement the hybrid method. The former uses randomly generated trees to form the initial population while the later uses sequential-addition to create quality starting trees. The detailed settings of the parameters (see Section 5.4) of these two programs are given below.

- Population size: $n_p = 20$.
- Offspring size: $n_o = 60$.
- Elite group size: $n_e = 5$.
- Tabu tenure = 7.
- Mutation rate: $p_m = 0.3$.
- $\epsilon = 2$.

- $T_{no} = 10$

Section 6.2 provides the design of experiments for our analysis. A comparison of the performance between the GATSpars program and the tabu-search-based program, tabupars, is given in Section 6.3. We assess the accuracy of the results produced by GATSpars and GATSpars* in Section 6.4. A discussion of the performance analysis is given in Section 6.5.

6.2 Design of Experiments

The 47-taxon DNA sequence data set mentioned in Chapter 3 is used to randomly create two groups of data. The first group contains 110 randomly generated data sets with 10 to 20 aligned DNA sequences. These data sets are the same as described in Section 4.1. The second group of data consists of 60 data sets. These data sets are created by randomly sampling m ($40 \leq m \leq 45$) sequences from the 47-taxon DNA sequence data set. For each m, 10 data sets are created.

The first group of data is used to assess the efficiency of the proposed hybrid method. To investigate the effectiveness of incorporating tabu search in the genetic algorithm, we created another C++ program, *GApars*, which implements a pure genetic algorithm. The design of the GApars program is exactly the same as GATSpars except that the population is not divided into *elite group* and *non-elite group* in GApars. As described in Section 4.2, the global optimal parsimony scores of the data sets in the first group are achieved by the *dnapenny* program, a branch-and-bound-based program. We run both of the GATSpars and GApars programs on these data sets but stop them when they achieve the global optimal parsimony scores. The number of parsimony evaluations are recorded for both programs. The results are compared with those derived from the tabupars program in Section 6.3.

The second group of data is used to assess the accuracy of the GATSpars and GATSpars* programs. We use the following programs: PAUP*, dnapars and GApars as the benchmarks. The PAUP* program is one of the most widely-used program for the maximum-parsimony phylogeny inference. It implements a hill-climbing heuristic search method and employs *tree bisection and recombination* (TBR) as the tree rearrangement method. The dnapars program uses the sequential-addition technique to construct the most parsimonious trees. During each step, a randomly selected taxon is added to the the tree by

being attached on the branch which results in the fewest evolutionary changes. If more than one trees are tied with the same parsimony scores, the *nearest neighbor interchange* (NNI) will be performed on these trees in order to find better results. The stopping conditions for the GATSpars, GATSpars* and GApars programs are the same. The searching process will be terminated if the best parsimony score does not improve in 100 iterations ($T_{stop} = 100$). PAUP* runs with its default setting while dnapars runs with its default setting and the *Jumble* option (random number seed = 1 and number of runs = 5). The results from these programs are compared in Section 6.4.

6.3 Assessing the Efficiency of the Hybrid Method

Table 6.1 demonstrates the average number of parsimony evaluations recorded when the programs, tabupars (TS), GApars (GA) and GATSpars (GATS), reach the global optimal parsimony scores. To know how efficient the GATSpars program is over the other two programs, we calculate the following ratios:

$$\text{ratio}^1 = \frac{\text{number of parsimony evaluations by GATSpars}}{\text{number of parsimony evaluations by tabupars}},$$

$$\text{ratio}^2 = \frac{\text{number of parsimony evaluations by GATSpars}}{\text{number of parsimony evaluations by GApars}}. \tag{6.1}$$

The log values of these two ratios are shown in the last two columns of Table 6.1. The lower the log-ratio values are, the better the GATSpars program outperforms the other two programs. Fig. 6.1 and 6.2 show the trend of $\log(\text{ratio}^1)$ and $\log(\text{ratio}^2)$, respectively.

The percentages of solution space searched by these three programs are defined as

$$100 \times \frac{\text{number of parsimony evaluations}}{\text{number of possible tree topologies}}. \tag{6.2}$$

Table 6.2 shows the average percentage of solution space searched by the tabupars, GApars and GATSpars programs. The trends of the average log-percentage values are demonstrated in Fig. 6.3.

The experimental results shown above indicate that both the GApars and GATSpars programs achieve the global optimal parsimony scores faster than the tabupars program. The trend shown in Fig. 6.1 also indicates that as the problem size increases, the better the GATSpars program outperforms the tabupars program. The GATSpars program performs

Table 6.1: The average number of parsimony evaluations before reaching the optimal solution.

number of taxa	TS	GA	GATS	Log-ratio^1	Log-ratio^2
10	4878	756	699	-1.942839857	-0.078390635
11	3720	790	790	-1.549446018	0
12	4574	1033	1110	-1.416028095	0.071892826
13	14059	1429	1484	-2.248521639	0.037766246
14	9131	1994	1802	-1.622778075	-0.101245513
15	40914	2042	2012	-3.012343082	-0.014800468
16	196995	5313	4232	-3.840503693	-0.227481955
17	84484	6479	3114	-3.300654128	-0.732658112
18	156351	4648	3591	-3.773672804	-0.258006307
19	453356	13396	4596	-4.591491377	-1.069769806
20	321753	6728	5778	-4.019726610	-0.152220322

Log-ratio^1 $= log(\frac{\text{GATS}}{\text{TS}})$ and Log-ratio^2 $= log(\frac{\text{GATS}}{\text{GA}})$.

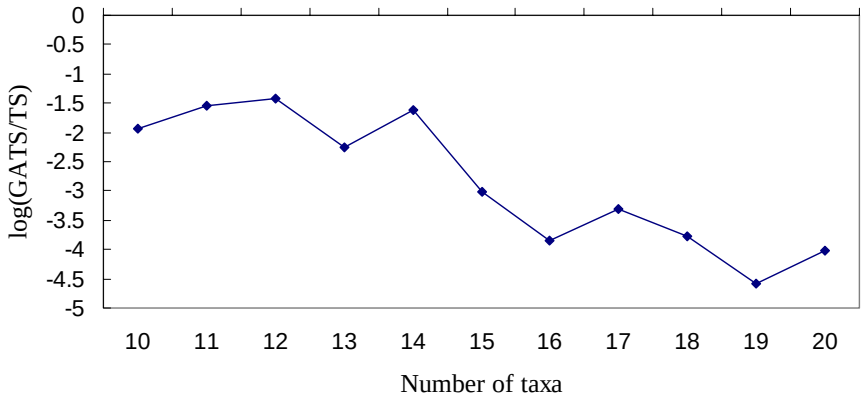


Figure 6.1: The trend of $\log(\text{ratio}^1)$ values.

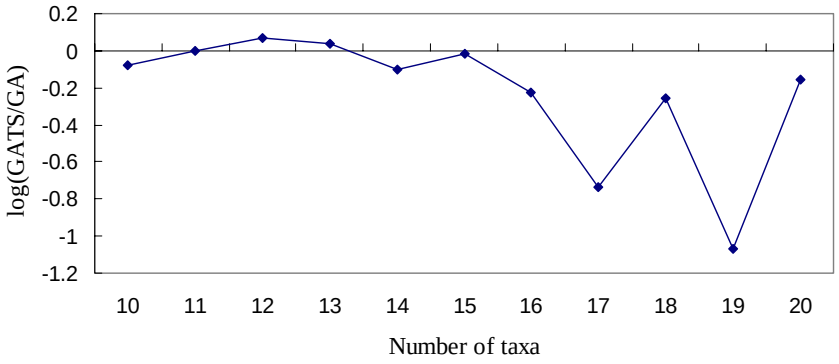


Figure 6.2: The trend of $\log(\text{ratio}^2)$ values.

Table 6.2: The average percentage of trees evaluated by these three programs before reaching the optimal maximum-parsimony trees.

number of taxa	TS	GA	GATS
10	1.41558E-02	2.19388E-05	2.02847E-05
11	5.68189E-04	1.20661E-06	1.20661E-06
12	3.32686E-05	7.51310E-08	8.07313E-08
13	4.44582E-06	4.51880E-09	4.69273E-09
14	1.15507E-07	2.52218E-10	2.27932E-10
15	1.91671E-08	9.56628E-12	9.42574E-12
16	3.18232E-09	8.58281E-13	6.83652E-13
17	4.40254E-11	3.37626E-14	1.62273E-14
18	2.46896E-12	7.33973E-16	5.67060E-16
19	2.04543E-13	6.04395E-17	2.07360E-17
20	3.92344E-15	8.20408E-19	7.04566E-19

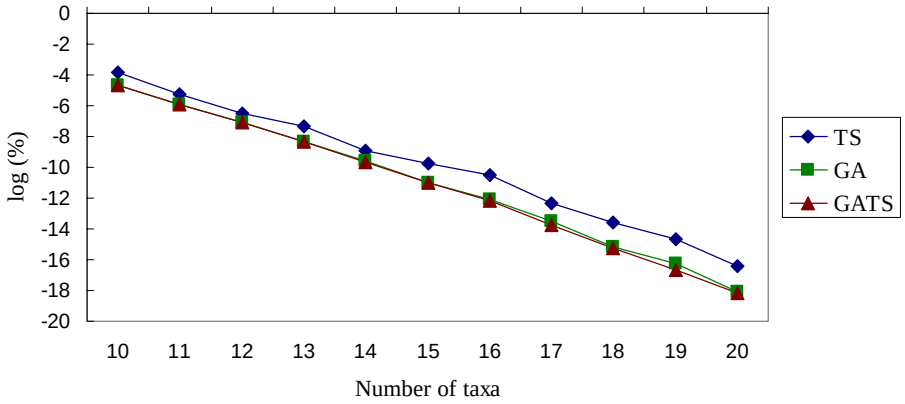


Figure 6.3: The trend of the log-percentage of solution space searched by these three programs.

slightly better than the GApars program but the difference is not very significant. Further investigation for the comparisons of the performance between the GATSpars and GApars programs are carried out in the next section.

6.4 Assessing the Accuracy of the Hybrid Method

We run the programs PAUP*, dnapars with the default settings, dnapars with the jumble option, GATSpars, GATSpars* and GApars on the data sets with number of taxa ranging from 40 to 45. Table 6.3 to Table 6.8 show the parsimony scores derived from these programs. We summarize the experimental results and assess the accuracy of the GATSpars and GATSpars* programs in subsections.

6.4.1 Assessing the Accuracy of GATSpars

The parsimony scores produced by the GATSpars program are compared with those from the PAUP*, dnapars with the default settings and dnapars with the jumble option programs. The comparisons are shown in Fig. 6.4, 6.5 and 6.6, respectively. The solid bars represent the percentage of instances the GATSpars program producing better parsimony scores than those produced by the other programs, while the striped bars represents the percentage of the worse instances. We can see from the results that PAUP* dominates all other programs. In more than 90% of instances, PAUP* achieves the best parsimony

scores among the results of all programs. In average, the GATSpars program can achieve the same or better parsimony scores than those from the PAUP* program in about 57% of instances. The dnapars program with the jumble option produces equally-good parsimony scores as those from GATSpars program, while its use of default settings performs slightly better. In average, the GATSpars program achieves the same or better parsimony scores than those from the dnapars program in about 63% of instances when the dnapars program is using default setting. This percentage increases to 68% when comparing the GATSpars program with the dnapars program using the jumble option.

6.4.2 Assessing the Accuracy of GATSpars*

The same comparisons are conducted for the GATSpars* program. The comparing results are shown in Fig. 6.7, 6.8 and 6.9. The solid bars represent the percentage of instances the GATSpars* program producing better parsimony scores than those produced by the other programs, while the striped bars represents the percentage of the worse instances. We can see that the GATSpars* program significantly improves in the accuracy of finding the most parsimonious trees. Although PAUP* still outperforms GATSpars*, the percentage of instances in which GATSpars* achieves the same or better parsimony scores than those from the PAUP* program increases to 74%. The GATSpars* program also performs better than the dnapars program with either its default settings or the jumble option. In average, the GATSpars* achieves the same or better parsimony scores than those derived from the dnapars program with the default settings and with the jumble option in about 79% and 86% of instances, respective. To demonstrate the improvement of the GATSpars* program, we summarize the percentage of instances the GATSpars* program achieves the best parsimony scores among the results produced by all the programs. We perform the same summarization on the GATSpars and GApars programs. The comparison between these three programs is shown in Fig. 6.10. The GATSpars* program has the best performance while the GApars program performs the worst.

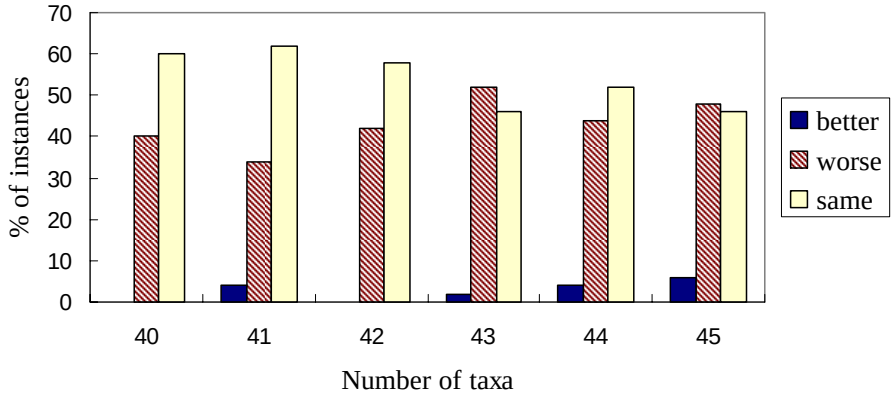


Figure 6.4: GATSpars versus PAUP*.

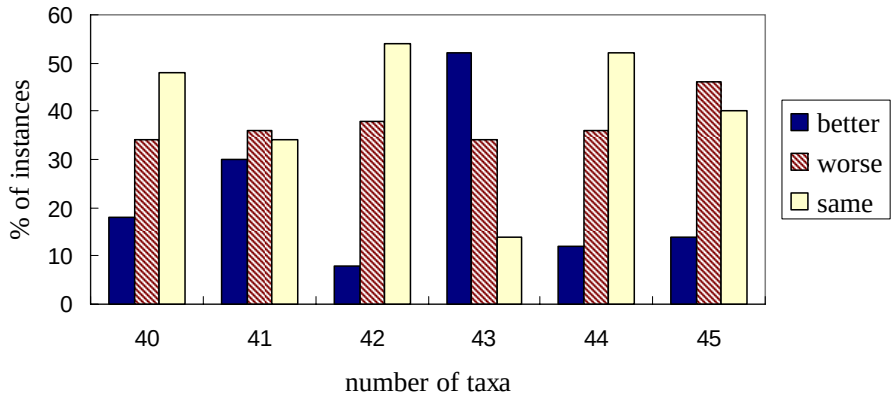


Figure 6.5: GATSpars versus dnapars (default).

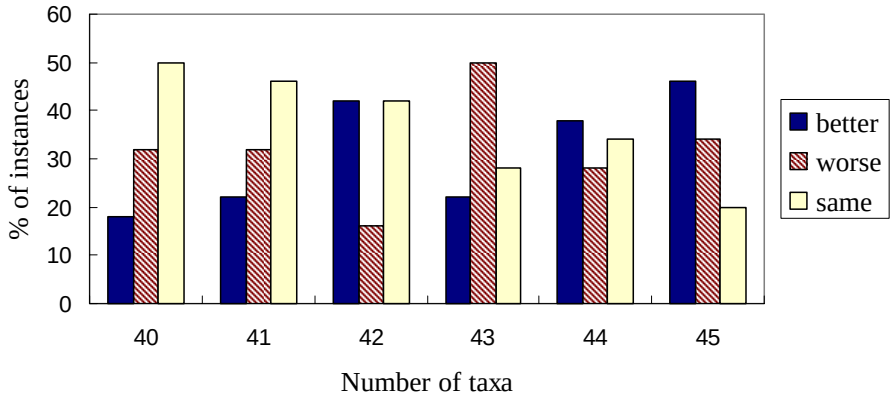


Figure 6.6: GATSpars versus dnapars (with jumble option).

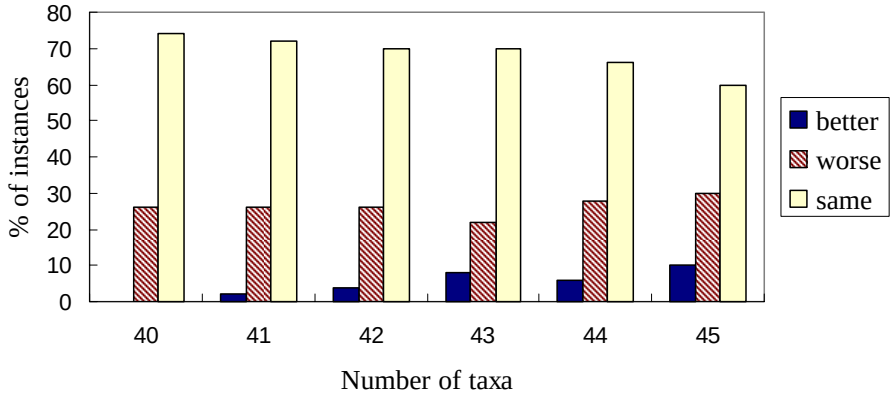


Figure 6.7: GATSpars* versus PAUP*.

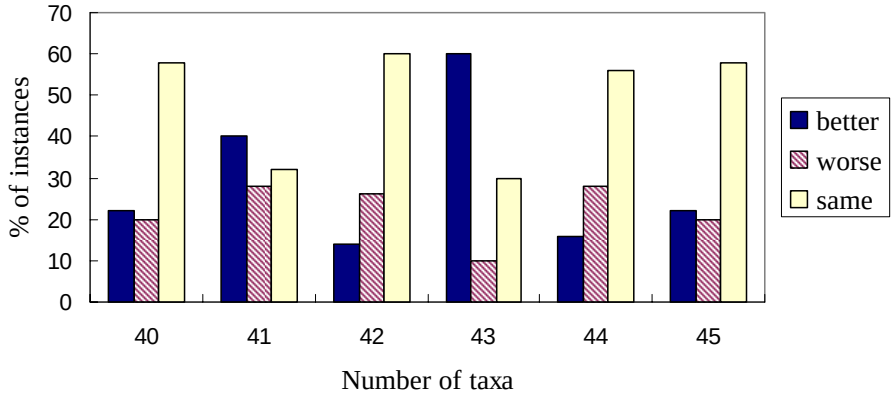


Figure 6.8: GATSpars* versus dnapars (default).

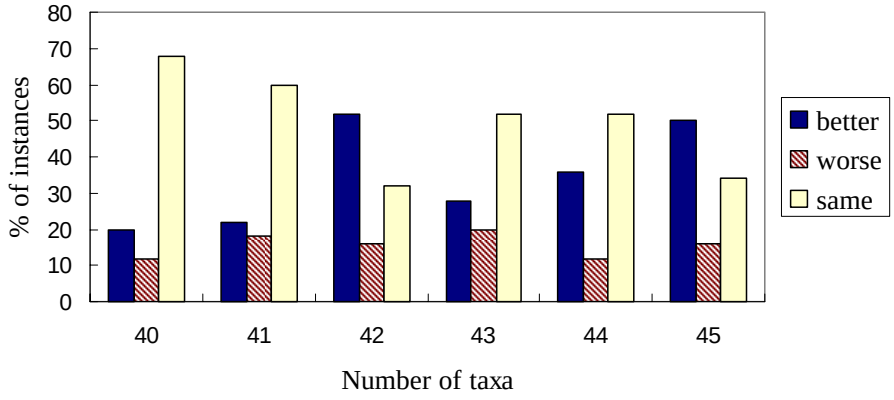


Figure 6.9: GATSpars* versus dnapars (with jumble option).

Table 6.3: The parsimony scores derived from each program on data sets with number of taxa = 40.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
PAUP*		948	974	927	1008	1007	969	977	940	956	978
dnapars(default)		949	974	927	1008	1009	969	977	940	956	979
dnapars(jumble)		948	980	927	1008	1007	969	977	940	956	987
GATSpars	Rep1	948	974	929	1008	1009	975	977	940	956	978
	Rep2	951	974	927	1008	1007	969	980	940	958	978
	Rep3	948	975	929	1008	1007	969	977	943	956	979
	Rep4	948	974	930	1021	1007	969	977	942	958	988
	Rep5	951	976	933	1010	1007	969	977	940	956	979
GATSpars*	Rep1	951	975	927	1009	1007	969	977	940	956	979
	Rep2	948	975	927	1008	1007	969	977	940	956	979
	Rep3	948	975	932	1010	1007	969	977	942	956	978
	Rep4	948	974	927	1008	1007	969	977	941	956	979
	Rep5	948	975	927	1008	1007	969	977	940	956	978
GApars	Rep1	958	976	928	1011	1017	973	1030	943	957	985
	Rep2	951	976	929	1008	1012	969	977	942	959	984
	Rep3	951	975	931	1011	1013	969	977	940	956	978
	Rep4	948	974	927	1018	1012	969	977	940	956	981
	Rep5	948	976	931	1011	1010	969	977	940	959	979

Table 6.4: The parsimony scores derived from each program on data sets with number of taxa = 41.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
PAUP*		996	1024	978	1031	980	969	1003	1002	934	1002
dnapars(default)		996	1024	980	1034	980	969	1002	1004	934	1007
dnapars(jumble)		996	1025	978	1031	980	969	1003	1002	936	1003
GATSpars	Rep1	996	1024	978	1031	980	969	1006	1002	934	1002
	Rep2	996	1027	978	1031	980	970	1003	1002	934	1002
	Rep3	996	1026	979	1031	982	969	1002	1002	934	1016
	Rep4	999	1025	981	1043	980	970	1003	1002	934	1002
	Rep5	996	1026	981	1031	982	969	1002	1009	937	1002
GATSpars*	Rep1	996	1024	978	1031	980	971	1003	1002	934	1002
	Rep2	999	1025	978	1031	981	970	1002	1002	934	1002
	Rep3	997	1024	978	1031	981	970	1003	1002	934	1003
	Rep4	996	1025	978	1031	980	969	1003	1002	934	1005
	Rep5	996	1025	978	1031	980	969	1004	1002	934	1002
GApars	Rep1	998	1025	980	1031	983	969	1012	1008	948	1005
	Rep2	1013	1032	978	1031	980	973	1043	1002	936	1017
	Rep3	996	1024	978	1031	980	969	1008	1004	934	1011
	Rep4	998	1025	981	1041	980	974	1003	1012	948	1005
	Rep5	997	1026	978	1031	981	984	1003	1006	934	1002

Table 6.5: The parsimony scores derived from each program on data sets with number of taxa = 42.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
PAUP*		997	1024	1025	1020	1013	1014	980	1040	980	1019
dnapars(default)		997	1024	1025	1020	1013	1014	980	1040	980	1023
dnapars(jumble)		1007	1027	1025	1020	1016	1014	983	1049	980	1019
GATSpars	Rep1	997	1028	1026	1020	1014	1014	980	1040	980	1019
	Rep2	1000	1027	1025	1020	1015	1014	984	1040	980	1019
	Rep3	1000	1027	1027	1020	1015	1014	980	1040	980	1019
	Rep4	1000	1024	1026	1022	1015	1014	980	1040	980	1021
	Rep5	999	1025	1025	1020	1014	1016	981	1040	980	1019
GATSpars*	Rep1	997	1024	1025	1022	1016	1014	980	1040	980	1019
	Rep2	999	1025	1026	1020	1013	1014	980	1040	980	1019
	Rep3	997	1024	1026	1019	1014	1016	980	1040	981	1019
	Rep4	999	1024	1025	1019	1015	1016	980	1040	980	1019
	Rep5	997	1024	1026	1020	1013	1014	980	1040	980	1019
GApars	Rep1	1000	1027	1025	1022	1016	1014	988	1043	981	1019
	Rep2	999	1024	1028	1019	1015	1014	987	1040	980	1021
	Rep3	997	1028	1026	1019	1015	1014	983	1040	980	1021
	Rep4	1006	1033	1026	1020	1014	1016	981	1043	980	1019
	Rep5	999	1027	1035	1019	1015	1024	983	1044	980	1019

Table 6.6: The parsimony scores derived from each program on data sets with number of taxa = 43.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
PAUP*		1012	1055	1013	1048	976	1021	1048	1020	1040	1056
dnapars(default)		1024	1057	1013	1060	976	1033	1052	1020	1045	1055
dnapars(jumble)		1012	1056	1013	1050	976	1021	1048	1020	1042	1055
GATSpars	Rep1	1015	1055	1016	1050	979	1021	1052	1023	1040	1055
	Rep2	1015	1055	1016	1048	976	1021	1048	1023	1040	1056
	Rep3	1018	1058	1014	1049	977	1021	1051	1020	1046	1056
	Rep4	1012	1055	1013	1048	976	1024	1053	1028	1040	1057
	Rep5	1012	1055	1016	1048	978	1022	1048	1020	1044	1056
GATSpars*	Rep1	1012	1055	1014	1049	1076	1021	1048	1020	1040	1055
	Rep2	1013	1055	1018	1049	1076	1021	1049	1020	1040	1055
	Rep3	1013	1055	1016	1048	1080	1021	1048	1020	1040	1056
	Rep4	1012	1055	1013	1048	1076	1021	1048	1020	1040	1055
	Rep5	1012	1055	1013	1051	1076	1025	1048	1020	1040	1055
GApars	Rep1	1015	1058	1035	1048	976	1024	1048	1024	1042	1058
	Rep2	1013	1059	1013	1049	985	1021	1050	1020	1040	1056
	Rep3	1015	1057	1023	1048	976	1021	1051	1023	1044	1056
	Rep4	1016	1057	1014	1049	978	1025	1048	1020	1042	1057
	Rep5	1013	1055	1024	1062	983	1025	1048	1029	1042	1056

Table 6.7: The parsimony scores derived from each program on data sets with number of taxa = 44.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
PAUP*		1053	1030	1028	1032	1074	1035	1047	1044	1059	1033
dnapars(default)		1053	1030	1028	1033	1074	1035	1047	1048	1059	1028
dnapars(jumble)		1053	1030	1030	1034	1076	1035	1047	1044	1063	1030
GATSpars	Rep1	1053	1030	1032	1032	1074	1039	1049	1046	1059	1036
	Rep2	1053	1030	1028	1032	1074	1035	1049	1047	1062	1033
	Rep3	1053	1030	1028	1033	1074	1035	1047	1044	1062	1028
	Rep4	1053	1030	1029	1033	1076	1039	1047	1044	1059	1044
	Rep5	1053	1036	1029	1036	1075	1035	1049	1048	1061	1028
GATSpars*	Rep1	1053	1030	1028	1032	1076	1035	1047	1044	1059	1028
	Rep2	1053	1030	1030	1034	1076	1039	1047	1044	1059	1031
	Rep3	1053	1032	1028	1032	1074	1035	1047	1044	1060	1031
	Rep4	1056	1030	1028	1035	1075	1035	1047	1044	1060	1028
	Rep5	1053	1030	1028	1032	1074	1035	1047	1044	1060	1028
GApars	Rep1	1053	1030	1030	1032	1075	1035	1048	1046	1072	1035
	Rep2	1053	1032	1038	1035	1075	1037	1050	1048	1063	1028
	Rep3	1058	1030	1028	1034	1078	1038	1059	1047	1060	1028
	Rep4	1054	1030	1028	1037	1075	1041	1048	1048	1060	1031
	Rep5	1053	1030	1029	1032	1082	1037	1048	1050	1066	1036

Table 6.8: The parsimony scores derived from each program on data sets with number of taxa = 45.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
PAUP*		1059	1068	1068	1037	1047	1008	1047	1048	1069	1031
dnapars(default)		1059	1069	1068	1037	1047	1008	1047	1049	1069	1031
dnapars(jumble)		1054	1071	1072	1050	1049	1008	1047	1048	1069	1034
GATSpars	Rep1	1066	1070	1068	1038	1049	1009	1050	1049	1069	1031
	Rep2	1056	1068	1068	1039	1047	1008	1047	1057	1072	1031
	Rep3	1059	1068	1071	1037	1047	1010	1053	1050	1069	1031
	Rep4	1054	1068	1068	1041	1047	1008	1050	1048	1074	1033
	Rep5	1054	1070	1068	1037	1047	1009	1049	1050	1070	1034
GATSpars*	Rep1	1054	1068	1068	1040	1047	1009	1049	1048	1069	1033
	Rep2	1054	1068	1068	1039	1047	1008	1047	1049	1069	1031
	Rep3	1054	1069	1068	1038	1047	1009	1047	1049	1069	1031
	Rep4	1054	1069	1069	1037	1047	1008	1047	1048	1073	1031
	Rep5	1055	1068	1068	1042	1047	1010	1047	1048	1069	1031
GApars	Rep1	1054	1069	1068	1040	1047	1008	1047	1050	1079	1031
	Rep2	1056	1068	1070	1037	1049	1012	1047	1050	1074	1031
	Rep3	1059	1084	1068	1037	1049	1008	1049	1051	1074	1033
	Rep4	1064	1070	1070	1039	1047	1008	1050	1050	1073	1037
	Rep5	1054	1070	1071	1041	1047	1008	1047	1051	1070	1031

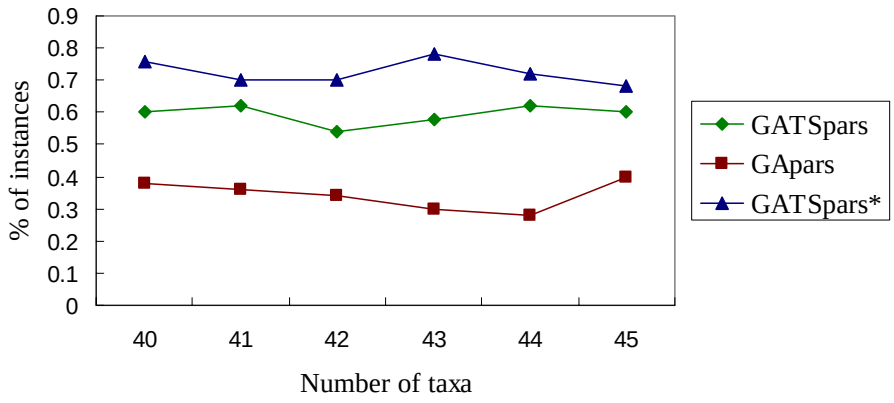


Figure 6.10: Percentage of achieving the best parsimony scores for the GATSpars, GATSpars* and GApars programs.

6.5 Discussion

From the results of Section 6.3, we have shown significant improvement for finding the maximum-parsimony trees with the hybrid method. The hybrid method achieves the global optimal solutions with much fewer number of parsimony evaluations. The results also indicate that the hybrid method performs slightly faster than the pure genetic algorithm program, although the differences of the performance between these two programs are not very significant when testing on the small-size data sets.

We then investigate the performance of the hybrid method on larger-size data sets. The results demonstrated in section 6.4 show stronger evidence that the hybrid method can produce much higher quality trees than the pure genetic algorithm. With the incorporation of the sequential-addition technique to generate starting trees, the hybrid method can produce better results than those produced by the dnapars program with its default settings and with the jumble option.

Although the proposed hybrid method is much more efficient than the proposed tabu search method, it is still relatively inefficient comparing to the PAUP* and dnapars programs. We can improve the hybrid method in several ways, for example, decreasing the chance the search process being trapped at the local optima or preprocessing the sequence data. We will have more discussion on future research in Chapter 9.

Chapter 7

Hybrid Method of Genetic Algorithm and Tabu Search for Maximum-likelihood Phylogeny Inference

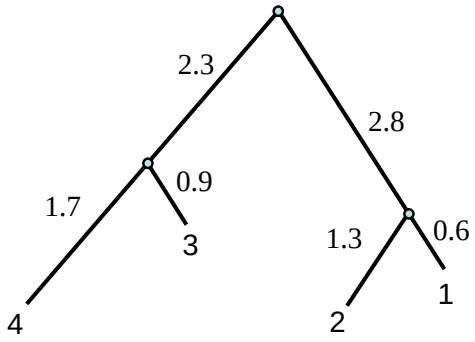
7.1 Introduction

In this chapter, we present a modified hybrid method which combines genetic algorithm and tabu search for maximum-likelihood phylogeny inference. The tree representation for the modified hybrid method is described in Section 7.2. The evaluation for the likelihood value of each tree is given in Section 7.3. A branch length optimization procedure is described in Section 7.4. In the last section, we present the detailed implementation of our hybrid search method for finding maximum-likelihood trees.

7.2 Tree Representation

The Newick format in Section 5.2 does not include the branch length information. One way to include the branch length information is to place branch length immediately after the group descended from that branch and separated by a full colon (See Fig. 7.1). The canonical form described in Section 5.2 is used to convert each tree into an one-to-one

corresponding Newick format. However, two issues rise during the process of converting a tree with branch length information into canonical form. First, the model of the evolution should be time-reversible so that re-rooting the tree will not change its likelihood value. We describe our model of evolution in Section 7.3. Second, when we re-position the root on the branch connecting to the smallest-index leaf, we have to decide how far away the root is from the smallest-index leaf. One way to deal with this issue is to place the root on the ascending node of the smallest-index leaf, i.e., the distance between the root and the ascending node of the smallest-index is zero (see Fig. 7.2). It is easy to see that the tree becomes an unrooted tree. Thus, the tree in Fig. 7.1 can be converted to its canonical form as shown in Fig. 7.3.



Newick format: ((4:1.7 , 3:0.9):2.3 , (2:1.3 , 1:0.6):2.8)

Figure 7.1: Newick format of a 4-leaf tree with branch length information.

7.3 Likelihood Computation and Model of Evolution

The computation of the likelihood for a two-leaf subtree is described in Chapter 1. In this section, the *pruning algorithm* for computing the likelihood for a whole tree is presented. The model of evolution used in our algorithm will also be described.

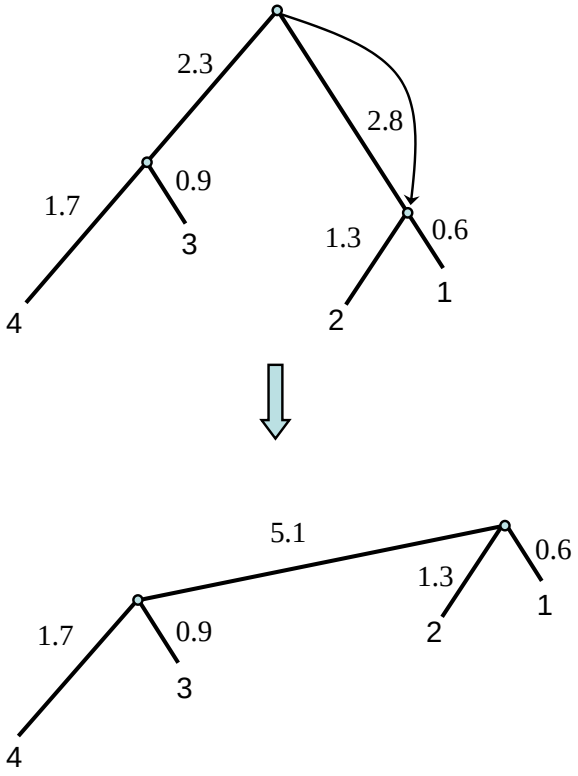
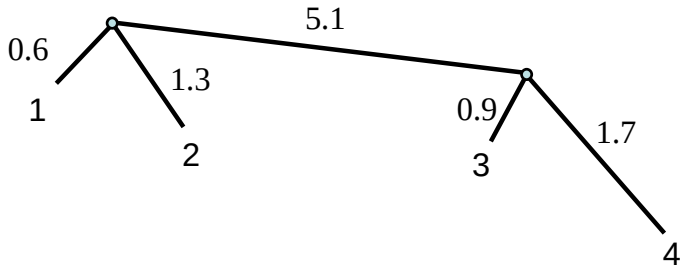


Figure 7.2: Re-root the tree in Fig. 7.1.



Newick format: (1:0.6 , 2:1.3 , (3: 0.9, 4:1.7):5.1)

Figure 7.3: Canonical form of the tree in Fig. 7.1.

7.3.1 Computing the Likelihood of a Phylogenetic Tree

Suppose we have a 5-leaf phylogenetic tree as shown in Fig. 7.4. Each leaf represents one of the aligned molecular sequences (e.g., DNA sequences) with m sites. There are two assumptions that are essential to computing the likelihoods:

1. Evolution in different sites on the given tree is independent.

2. Evolution in different branches is independent.

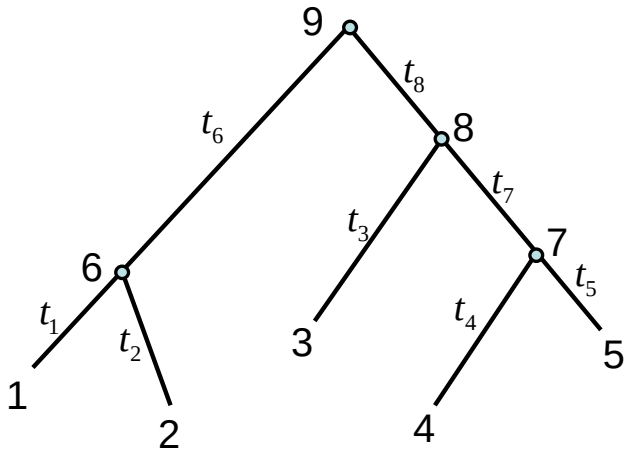


Figure 7.4: An example for explaining the likelihood calculation.

From the first assumption, we can decompose the likelihood computation into a product of the likelihood for each site as follow:

$$L = P(\text{data}|\text{tree}) = \prod_{i=1}^{m} P(\text{data}^i|\text{tree}) = \prod_{i=1}^{m} L^i, \tag{7.1}$$

where data^i is the sequence at the ith site and L^i is the likelihood of the tree considering only data^i. With this assumption, we only need to know how to calculate the likelihood for at a single site.

Now, suppose the sequence data of each leaf in Fig. 7.4 at site i are $s_1^i, s_2^i, \ldots, s_5^i$. Then the likelihood of the tree at site i can be stated as

$$L^i = \sum_{s_9} \sum_{s_8} \sum_{s_7} \sum_{s_6} P(s_1^i, \ldots, s_5^i, s_6, \ldots, s_9|\text{tree}), \tag{7.2}$$

where s_k is the hypothetic sequence data for the interior node k. Since its value is not known, we have to sum up the probabilities for all possible sequence values.

The second assumption allows us to break down the probability function of Eq. 7.2 into a product of conditional probabilities for each branch:

$$\begin{aligned} &P(s_1^i, \ldots, s_5^i, s_6, \ldots, s_9|\text{tree}) = \\ &P(s_9) \cdot P(s_6|s_9, t_6) \cdot P(s_8|s_9, t_8) \cdot P(s_1^i|s_6, t_1) \cdot P(s_2^i|s_6, t_2) \cdot \\ &\qquad P(s_3^i|s_8, t_3) \cdot P(s_7|s_8, t_7) \cdot P(s_4^i|s_7, t_4) \cdot P(s_5^i|s_7, t_5), \end{aligned} \tag{7.3}$$

where $P(s_9)$ is the equilibrium probability of the base s_9 (for example, in Jukes-Cantor's model, $\pi_A = \pi_C = \pi_T = \pi_G = \frac{1}{4}$) and $P(s_j|s_i, t_j)$ is the transition probability of state s_i evolves to state s_j through branch length t_j. The computation of the transition probability will be discussed in the next section.

The problem with the above calculation is that we have to sum up the probability terms for all possible sequence values of the interior nodes. Suppose we are dealing with DNA sequence data in this case. There will be $4^{(5-1)}$ terms. But if the number of leaves of the tree increases to 20, the number of terms is 4^{19}. To resolve this problem, Felsenstein introduced a dynamic-programming-based algorithm, called *pruning algorithm*, to efficiently calculate the likelihood [16]. Before we get into the details of this algorithm, we have to introduce the *conditional likelihood* of a subtree. Let $L_k^i(s)$ be the conditional likelihood of the subtree rooted at node k which has sequence value s at site i. For example, the conditional likelihood of the subtree rooted at node 6 in Fig. 7.4 can be stated as

$$L_6^i(s_6) = \left(\sum_{s_1} P(s_1|s_6, t_1) L_1^i(s_1) \right) \left(\sum_{s_2} P(s_2|s_6, t_2) L_2^i(s_2) \right). \tag{7.4}$$

Note that nodes 1 and 2 are leaves of the tree and their conditional likelihood can be defined as

$$L_k^i(s_k) = \begin{cases} 1, & \text{if } s_k = s_k^i \text{ ;} \\ 0, & \text{otherwise.} \end{cases} \tag{7.5}$$

The root of the tree is node 9 and the likelihood of this tree at site i is

$$L^i = \sum_{s_9} P(s_9) L_9^i(s_9). \tag{7.6}$$

The detailed procedure of the pruning algorithm is described in Algorithm 7.1.

Algorithm 7.1 The procedure of the pruning algorithm

Initialization:

Set Log-likelihood = 0.

Set site: $i = 1$.

Do while ($i \leq$ number of sites)

$\forall s_{root}$, call function $L^i_{root}(s_{root}) = L(i, root, s_{root})$.

Log-likelihood = Log-likelihood + $\log \left(\sum_{s_{root}} P(s_{root}) \cdot L^i_{root}(s_{root}) \right)$.

$i = i + 1$.

End

Output Log-likelihood.

Function $L(site, current, s_{current})$:

Let $\ell = current \rightarrow left\_child$.

Let $r = current \rightarrow right\_child$.

If ℓ is a leaf of the tree

$$L^{site}_{\ell}(s_{\ell}) = \begin{cases} 1, & \text{if } s_{\ell} = s^{site}_{\ell} \\ 0, & \text{otherwise.} \end{cases};$$

Else

$\forall s_{\ell}$, call function $L^{site}_{\ell}(s_{\ell}) = L(site, \ell, s_{\ell})$.

End

If r is a leaf of the tree

$$L^{site}_{r}(s_{r}) = \begin{cases} 1, & \text{if } s_{r} = s^{site}_{r} \\ 0, & \text{otherwise.} \end{cases};$$

Else

$\forall s_r$, call function $L^{site}_{r}(s_r) = L(site, r, s_r)$.

End

$$L^{site}_{current}(s_{current}) = \left(\sum_{s_{\ell}} P(s_{\ell}|s_{current}, t_{\ell}) \cdot L^{site}_{\ell}(s_{\ell}) \right) \left(\sum_{s_r} P(s_r|s_{current}, t_r) \cdot L^{site}_{r}(s_r) \right).$$

Return $L^{site}_{current}(s_{current})$.

7.3.2 Model of Evolution

To compute the transition probability $P(s_{k1}|s_{k2}, t)$ mentioned in the previous subsection, we have to define the model of evolution. In our hybrid method for searching the maximum-likelihood trees, we adopt F84 model as our evolution model. The F84 model was proposed by Felsenstein and is used in his phylogeny package, PHYLIP [18]. It is a time-reversible model, i.e., $P(s_{k1})P(s_{k1}|s_{k2}, t) = P(s_{k2})P(s_{k2}|s_{k1}, t)$. The transition rate matrix can be described as below.

Table 7.1: Transition matrix of F84 model.

	A	G	C	T
A	-	$\frac{\alpha\pi_G}{\pi_R} + \beta\pi_G$	$\beta\pi_C$	$\beta\pi_T$
G	$\frac{\alpha\pi_A}{\pi_R} + \beta\pi_A$	-	$\beta\pi_C$	$\beta\pi_T$
C	$\beta\pi_A$	$\beta\pi_G$	-	$\frac{\alpha\pi_T}{\pi_Y} + \beta\pi_T$
T	$\beta\pi_A$	$\beta\pi_G$	$\frac{\alpha\pi_C}{\pi_Y} + \beta\pi_C$	-

π_A, π_T, π_C and π_G are the equilibrium probabilities of four DNA characters. $\pi_R = \pi_A + \pi_G$ and $\pi_Y = \pi_C + \pi_T$ represent the pool of *purine* and *pyrimidine*, respectively. The parameters α and β are defined as

$$\alpha = \frac{\pi_R\pi_y R - \pi_A\pi_G - \pi_C\pi_T}{2(1+R)(\pi_Y\pi_A\pi_G + \pi_R\pi_C\pi_T)}, \tag{7.7}$$

$$\beta = \frac{1}{2\pi_R\pi_Y(1+R)}, \tag{7.8}$$

where

$$R = \frac{\text{total transition rate}}{\text{total transversion rate}}.$$

The transition probabilities between each DNA state then can be computed as in Table 7.2.

7.4 Branch Length Optimization

The optimization of branch lengths is implemented based on the estimation of branch length under site-homogeneous model described by Yang [60]. The basic concept goes as follows. Since the evolution model used to evaluate the likelihood of trees is time-reversible, the location of the root does not affect the likelihood of a tree. Suppose we are

Table 7.2: Transition probability of F84 model.

$p(A|A,t) = e^{-(\alpha+\beta)t} + e^{-\beta t}(1-e^{-\alpha t})\frac{\pi_A}{\pi_R} + (1-e^{-\beta t})\pi_A.$

$p(G|A,t) = e^{-\beta t}(1-e^{-\alpha t})\frac{\pi_G}{\pi_R} + (1-e^{-\beta t})\pi_G.$

$p(C|A,t) = (1-e^{-\beta t})\pi_C.$

$p(T|A,t) = (1-e^{-\beta t})\pi_T.$

$p(A|G,t) = e^{-\beta t}(1-e^{-\alpha t})\frac{\pi_A}{\pi_R} + (1-e^{-\beta t})\pi_A.$

$p(G|G,t) = e^{-(\alpha+\beta)t} + e^{-\beta t}(1-e^{-\alpha t})\frac{\pi_G}{\pi_R} + (1-e^{-\beta t})\pi_G.$

$p(C|G,t) = (1-e^{-\beta t})\pi_C.$

$p(T|G,t) = (1-e^{-\beta t})\pi_T.$

$p(A|C,t) = (1-e^{-\beta t})\pi_A.$

$p(G|C,t) = (1-e^{-\beta t})\pi_G.$

$p(C|C,t) = e^{-(\alpha+\beta)t} + e^{-\beta t}(1-e^{-\alpha t})\frac{\pi_C}{\pi_Y} + (1-e^{-\beta t})\pi_C.$

$p(T|C,t) = e^{-\beta t}(1-e^{-\alpha t})\frac{\pi_T}{\pi_Y} + (1-e^{-\beta t})\pi_T.$

$p(A|T,t) = (1-e^{-\beta t})\pi_A.$

$p(G|T,t) = (1-e^{-\beta t})\pi_G.$

$p(C|T,t) = e^{-\beta t}(1-e^{-\alpha t})\frac{\pi_C}{\pi_Y} + (1-e^{-\beta t})\pi_C.$

$p(T|T,t) = e^{-(\alpha+\beta)t} + e^{-\beta t}(1-e^{-\alpha t})\frac{\pi_T}{\pi_Y} + (1-e^{-\beta t})\pi_T.$

trying to optimize the length of the branch between nodes m and n on the tree of Fig. 7.5 (a). We can place the root on the target branch and let the length of the branch between the root and node m be zero. The likelihood of this tree for a particular site, i, of sequences can be computed as

$$L^i = \sum_{s_m} \sum_{s_n} \pi_{s_m} L_m^i(s_m) L_n^i(s_n) p(s_n|s_m, t), \tag{7.9}$$

and the log-likelihood of this tree among all sites of sequences becomes

$$L = \sum_i \log L^i, \tag{7.10}$$

where s_m and s_n are the sequence characters (e.g., A, T, C or G as in DNA sequences) at site i, π_{s_m} is the equilibrium probability of drawing a character s_m, L_m^i and L_n^i are the conditional likelihood of the subtrees descending from nodes m and n, respectively, and $p(s_n|s_m, t)$ is the transition probability of state s_m evolving to state s_n through branch length t. It can be seen that $p(s_n|s_m, t)$ is the only term in equation (7.9) that contains t. Hence, the first and second derivatives of equation (7.9) with respect to t can be written as

$$\begin{aligned} \frac{\partial L^i}{\partial t} &= \textstyle\sum_{s_m} \sum_{s_n} \pi_{s_m} L_m^i(s_m) L_n^i(s_n) \frac{\partial p(s_n|s_m,t)}{\partial t}, \\ \frac{\partial^2 L^i}{\partial t^2} &= \textstyle\sum_{s_m} \sum_{s_n} \pi_{s_m} L_m^i(s_m) L_n^i(s_n) \frac{\partial^2 p(s_n|s_m,t)}{\partial t^2}. \end{aligned} \tag{7.11}$$

Therefore, from equations (7.10) and (7.11), we can compute the first and second derivatives of equation (7.10) with respect to t as

$$\begin{aligned} L' &= \sum_i \frac{(L^i)'}{L^i}, \\ L'' &= \sum_i \frac{L^i \cdot (L^i)'' - ((L^i)')^2}{(L^i)^2}. \end{aligned} \tag{7.12}$$

Once we have the information of the first and second derivatives, we can use Newton-Raphson method to update the branch length t iteratively according to the following formula:

$$t^{(k+1)} = t^{(k)} + \alpha \cdot \frac{L'}{L''}, \tag{7.13}$$

where α is the step length parameter. Usually, we set its value equal to 1. However, when $t^{(k+1)}$ produces a worse likelihood value, we reduce the value of α, say by half, until the new

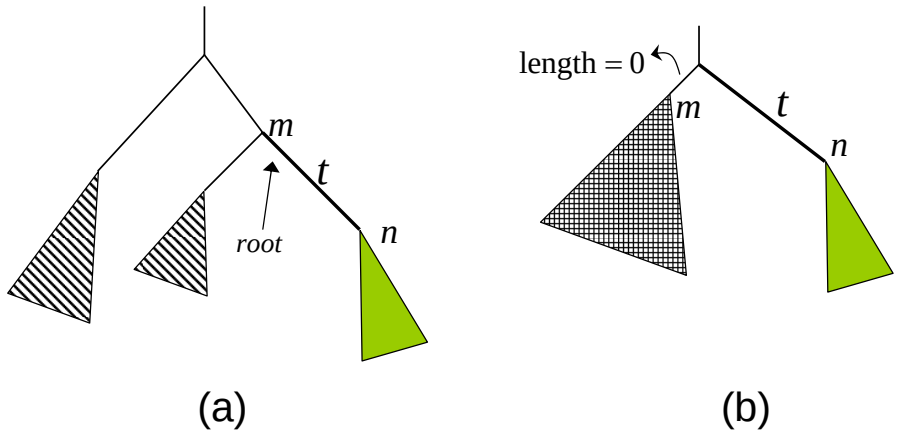


Figure 7.5: (a) Original topology. (b) Re-root the tree by placing the root on the branch between nodes m and n.

likelihood value is not worse. Therefore, this branch optimization process is nondecreasing. Note that Eq. (7.13) only works when $L^{''} < 0$ (concave). When $L^{''} > 0$, we apply a simple line search with the first-derivative information. The optimization of a single branch stops when no significant improvement observed during iterations, or the value of $|L^{'}|$ becomes too small. More specifically, we stop the single branch optimization when $|L_{\text{new}} - L_{\text{old}}| < \epsilon$ or $|L^{'}| < \delta$.

We repeat the optimization procedure on each branch of the tree iteratively until all branches are optimized. Note that this process usually needs to be carried out for more than one round since after one branch is optimized, other branches may need to be re-optimized. Therefore, we continue this optimization process for several rounds until no significant improvement is observed, i.e., $|L_{\text{new}} - L_{\text{old}}| < \epsilon$. Also note that the conditional likelihoods of the subtrees need to be updated every time we re-position the root. Fortunately, we can minimize the number of re-evaluating the conditional likelihoods by applying a dynamic-programming-based method. The detailed procedure of this method is described in Algorithm 7.2.

Algorithm 7.2 The procedure of the branch lengths optimization

Initialization:
Start from the root of the original tree.
Evaluate the conditional likelihood, $L_k^i(s_k)$, for each site i, node k and state s_k.
Set $\mathscr{R}$ = original root.
Call function: $Opt\_Branch(\mathscr{R})$.

If $|L_{\text{new}} - L_{\text{old}}| < \epsilon$
STOP.
Else
Set $\mathscr{R}$ = original root.
Call function: $Opt\_Branch(\mathscr{R})$.
End

Function $Opt\_Branch(\mathscr{R})$:
Update $L_{\mathscr{R}}^i(s_{\mathscr{R}}), \forall i, s_{\mathscr{R}}$.
Let $\ell = \mathscr{R} \rightarrow$ left_child.
Let $r = \mathscr{R} \rightarrow$ right_child.

Find t_ℓ^* = Newton-Raphson(t_ℓ, L_ℓ, L_r).
Let $\ell \rightarrow$ left_child $= \mathscr{R}$ and $\mathscr{R} = \ell$.
Call function $Opt\_Branch(\mathscr{R})$.

Restore $\mathscr{R}$ and ℓ.
Update $L_{\mathscr{R}}^i(s_{\mathscr{R}}), \forall i, s_{\mathscr{R}}$.

Find t_r^* = Newton-Raphson(t_r, L_ℓ, L_r).
Let $r \rightarrow$ right_child $= \mathscr{R}$ and $\mathscr{R} = r$.
Call function $Opt\_Branch(\mathscr{R})$.

Restore $\mathscr{R}$ and r.
Update $L_{\mathscr{R}}^i(s_{\mathscr{R}}), \forall i, s_{\mathscr{R}}$.

7.5 Hybrid Method of Genetic Algorithm and Tabu Search for Finding Maximum-likelihood Trees

The design of the hybrid method of genetic algorithm and tabu search for the maximum-likelihood phylogeny inference is similar to the one described in Section 5.4. The population is divided into two groups: elite group and non-elite group. We use the same crossover and mutation operators as described in Chapter 5. Also, the tabu list stores tree topologies only. Thus, trees with the same topologies as those on the tabu list but with different branch lengths are considered tabooed.

After the offspring are created either by crossover or mutation, the branch length of each offspring is optimized and the fitness score of each offspring is evaluated based on the likelihood computation described in the previous sections. The branch length optimization process and the likelihood evaluation are computationally expensive. Especially, most of the computation time is spent on optimizing the branch lengths of the non-optimal tree topologies. In our hybrid method, there are two ways to deal with this issue. First, we loosen the stopping conditions for the branch length optimization, i.e., set larger values for ϵ and δ. During the search process, the best n_B trees are recorded. At the end of the search process, the stopping conditions are tightened and the branch lengths of the best trees are re-optimized.

The second way to save the computation time is to use the parsimony values of the offspring. Fig. 7.6 shows the relationship between the likelihood and the parsimony score. 10,000 phylogenetic trees were randomly generated for the 47-taxon data described in Section 4.1. Both the likelihood and the parsimony score are evaluated for each tree. The Pearson correlation coefficient between the likelihood values and parsimony scores is -0.96332. This tells us that the parsimony score of a tree is a good indicator of how good or bad its likelihood value could be. As a result, we set a threshold to filter out *bad* offspring before we optimize their branch lengths. Suppose the parsimony score of tree t_i is p_{t_i} and the best parsimony score we have is p^*. Then, t_i is discarded before optimizing its branch lengths and evaluating its likelihood value if

$$p_{t_i} - p^* > \varepsilon. \tag{7.14}$$

The value of ε can be changed during the search process. For example, if there are too

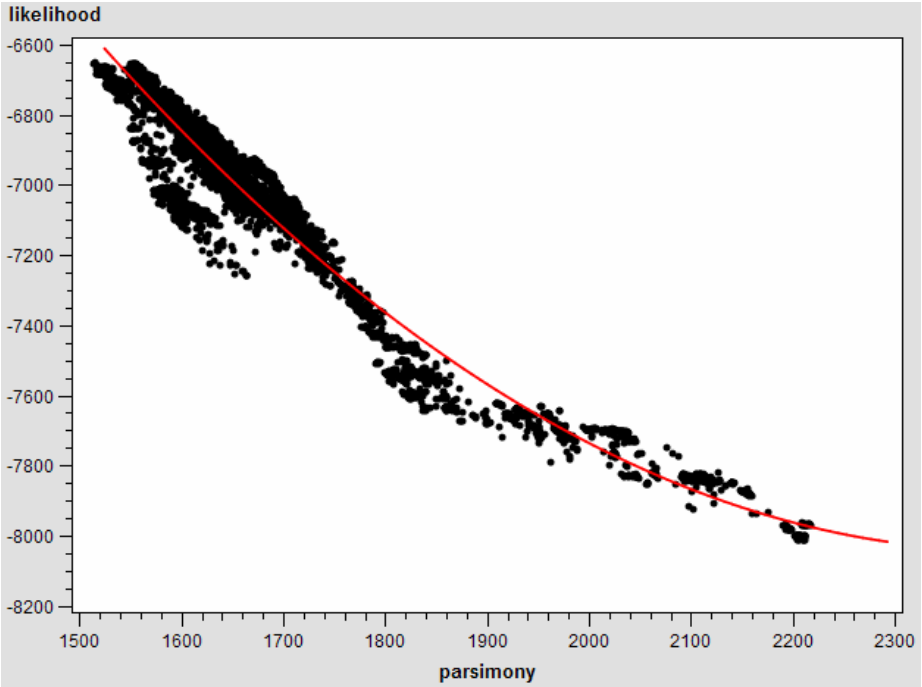


Figure 7.6: Likelihood versus parsimony.

many local optimal parsimony trees around the most parsimonious trees, the value of ε is decreased to allow us narrowing down the search targets. Also, if the search process is trapped in a local optimum, the value of ε is increased to allow us investigating the trees away from the most parsimonious trees.

The detailed procedure of the hybrid method for the maximum-likelihood phylogeny inference is described in Algorithm 7.3.

Algorithm 7.3 Procedure for the hybrid method of GA and TS for maximum-likelihood phylogeny inference.

Initialize:
Generate an initial population: P_0, where $\|P_0\| = n_p$.
Set the stopping conditions, ϵ and δ, for branch length optimization.

Optimize the branch length and evaluate the likelihood of the initial population.
Rank all trees in the population based on the likelihood value
and find the best trees t_0^*.

Update the best trees: $t^* = t_0^*$.
Update the best likelihood value: $L^* = \mathscr{L}(t_0^*)$,
where $\mathscr{L}$ is the function for likelihood evaluation.
Store the best n_B trees in t_B^*.

Evaluate the parsimony score of the initial population.
Update the best parsimony score: p^*.

Select the top n_e trees into the elite group, $\mathscr{E}_0$.
Put the rest of the population in the non-elite group, $\mathscr{N}_0$.

Set *tabu list* $= \mathscr{E}_0$.
Set iteration counter: $i = 0$.

Do while (stopping conditions not met)
Set offspring: $C_i = \phi, M_i = \phi$.
Crossover on elite group: $\forall x, y \in \mathscr{E}_i$ and $x \neq y$, $C_i = C_i \cup Cr(x, y)$.
Mutation on elite group: $\forall x \in \mathscr{N}_i, M_i = M_i \cup Mu^1(x)$.

Do while (number of offspring $< n_o$)
Crossover on non-elite group: $Rand(x, y) \in \mathscr{N}_i$ and $x \neq y, C_i = C_i \cup Cr(x, y)$.
Mutation on non-elite group: $\forall x \in \mathscr{N}_i$, if $RN < p_m, M_i = M_i \cup Mu^1(x)$,
where RN is a random number generated from random number generator.
End

If no improvement for T_{no} iterations
All-branches-mutation on best trees: $\forall x \in t^*, M_i = M_i \cup Mu^2(x)$.
$\varepsilon = \varepsilon + \Delta$.
End

Algorithm 7.3 Cont'd

Set $O_i = C_i \cup M_i$.

$\forall x \in O_i$ evaluate its parsimony score p_x
If $p_x - p^* > \varepsilon$
  $O_i = O_i - \{x\}$.
Else
  Optimize the branch length of x and evaluate $\mathscr{L}(x)$.
End

If $\|O_i\| < n_o$
  $\varepsilon = \varepsilon + \Delta$.
Else if $\|O_i\| > 2 * n_o$
  $\varepsilon = \varepsilon - \Delta$.

Create the ranking for the offspring: $R = Rank(O_i)$.
Find the best trees $t_i^* \in O_i$.

If $\mathscr{L}(t_i^*) < L^*$
  $L^* = \mathscr{L}(t_i^*)$ and $t^* = t_i^*$.
End

If $\exists\ x \in O_i$ and $y \in t_B^*$ such that $\mathscr{L}(x) > \mathscr{L}(y)$
  $t_B^* = t_B^* + \{x\} - \{y\}$.
End

Set new elite group and non-elite group: $\mathscr{E}_{i+1} = \phi, \mathscr{N}_{i+1} = \phi$.
Set $k = 1$.

Do while $(\|\mathscr{E}_{i+1}\| < n_e)$
  If $R(k) \notin$ *tabu list*
    $\mathscr{E}_{i+1} = \mathscr{E}_{i+1} \cup R(k)$.
    $R = R - \{R(k)\}$.
  End
  $k = k + 1$.
End

Algorithm 7.3 Cont'd

Reset $k = 1$.

Do while $\left(\|\mathscr{N}_{i+1}\| < (n_p - n_e)\right)$
If $RN < 1 - \frac{\mathscr{L}\left(R(k)\right) - L^* + \epsilon}{\mathscr{L}\left(R(k)\right)}$
$\mathscr{N}_{i+1} = \mathscr{N}_{i+1} \cup R(k)$.
End
$k = k + 1$.
End

Update *tabu list*.
Set $i = i + 1$.
End

Tighten the stopping conditions, ϵ and δ, for branch length optimization.
Re-optimize the branch lengths of the trees in t_B^*.
Output t_B^* and $\mathscr{L}(t_B^*)$.

Chapter 8

Performance of the Hybrid Method for Maximum-likelihood Phylogeny Inference

8.1 Introduction

In this chapter, we analyze the performance of the proposed hybrid method for finding the maximum-likelihood trees. We create a C++ program, *GATSml*, to implement the hybrid method for the maximum-likelihood phylogeny inference. This program also employs the sequential-addition technique to generate the starting trees. The detailed settings of the parameters (see Section 7.5) are described as below:

- Population size: $n_p = 20$.
- Offspring size: $n_o = 60$.
- Elite group size: $n_e = 5$.
- Tabu tenure = 7.
- Mutation rate: $p_m = 0.3$.
- $T_{no} = 10$.
- $T_{stop} = 100$.

- Threshold for parsimony score: $\varepsilon = 20$.
- Increment or decrement of parsimony threshold: $\Delta = 5$.
- Initial stopping conditions for branch length optimization: $\epsilon = 0.001, \delta = 0.5$.
- Tightened stopping conditions for branch length optimization: $\epsilon = 1.0E{-}6, \delta = 0.001$.

Section 8.2 provides the design of experiments for this analysis. Comparisons of the results between the GATSml and dnaml programs are presented in Section 8.3. A discussion of the performance analysis is found in Section 8.4.

8.2 Design of Experiments

The 110 data sets randomly created from the 47-taxon DNA sequence data are used again here for the the performance analysis of the GATSml program. The program *dnaml* in Felsenstien's PHYLIP package is used as the benchmark. The dnaml program uses F84 model as described in Section 7.3 to evaluate the likelihood of phylogenetic trees. Thus, the results derived from the GATSml and dnaml programs are comparable. Dnaml also employs sequential-addition to construct the maximum-likelihood trees. The NNI tree rearrangement method are performed to find better trees if there are more than one tree tied with the same likelihood values. We run the GATSml program on the 110 data sets with the settings of parameters described in the previous section. The results are compared with those derived from the default-setting dnaml program.

In order to assess the capability of the GATSml program in solving larger data, we run this program on the 47-taxon data set. The GATSml program is run under two different stopping conditions. The first one is as described in the previous section ($T_{stop} = 100$) while the second one is tightened ($T_{stop} = 200$). The results are compared with the dnaml program with the default settings and with the jumble option (random number seed = 1, number of runs = 5).

8.3 Assess the Performance of the Hybrid Method

We run both the GATSml and dnaml programs on the 110 data sets with number of involving taxa ranging from 10 to 20. The output likelihood values are recorded. These

results are shown in Tables 8.2 to 8.12. The summary of the comparisons is demonstrated in Fig. 8.1. The solid bars represent the number of instances the GATSml program producing better likelihood values than those produced by the dnaml program, while the striped bars represent the number of the worse instances. The GATSml program performs slightly better than the dnaml program but the difference is not significant. In average, the GATSml program performs better in 43% of instances, the same in 17% of instances and worse in 40% of instances. When reaching the reported best likelihood value, the number of likelihood evaluations is recorded and the trend of its value is shown in Fig. 8.2. When the data size becomes larger, the number of likelihood evaluations required to find the maximum-likelihood trees becomes much larger. A tighter stopping condition (larger value for T_{stop}) may be necessary.

We run the GATSml program on the 47-taxon data set in order to investigate the capability of the GATSml program in finding maximum-likelihood trees on larger data sets. We run the GATSml program with two different stopping conditions ($T_{stop} = 100$ and $T_{stop} = 200$). As shown in Table 8.1, the results are compared with those derived from the dnaml program with the default settings and with the jumble option. With either $T_{stop} = 100$ or $T_{stop} = 200$, the GATSml program produces better results than those produced by the default-setting dnaml program in all the instances. The GATSml program with $T_{stop} = 100$ has equally good performance as the dnaml program with the jumble option. The performance of the GATSml program is slightly improved when set $T_{stop} = 200$. The output maximum-likelihood tree is demonstrated in Fig. 8.3.

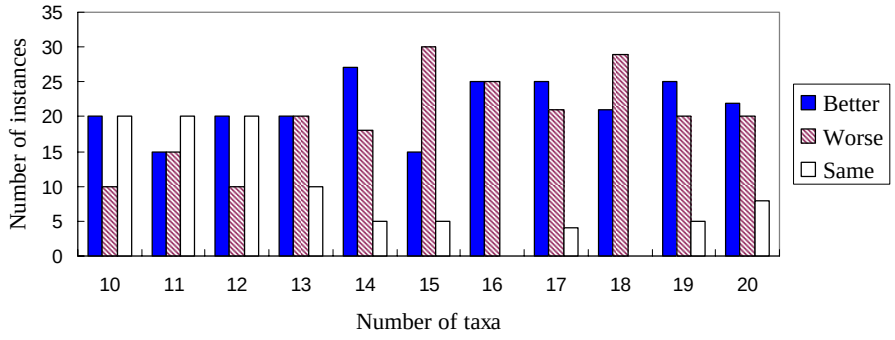


Figure 8.1: Comparisons of the performance between the GATSml and dnaml programs.

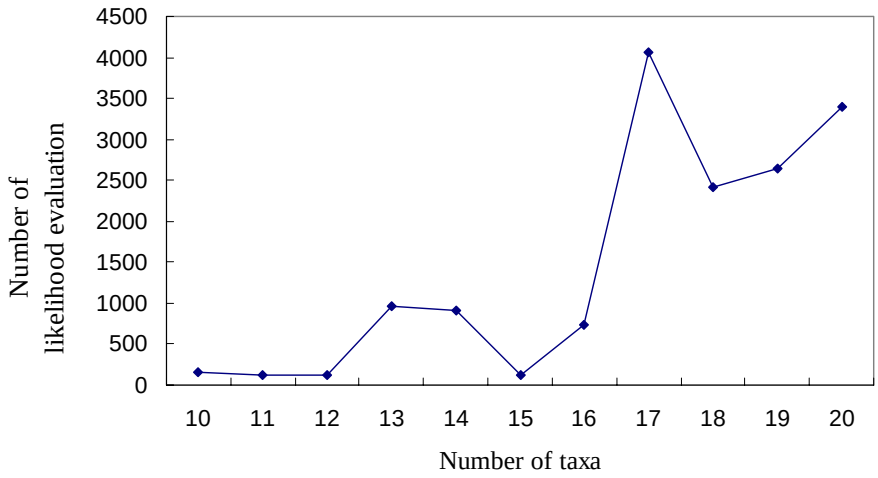


Figure 8.2: The trend of the number of likelihood evaluations required when reaching the reported best likelihood scores.

Table 8.1: The comparisons of the performance between the GATSml and dnaml programs on the 47-taxon data set.

Program	Replication	
GATSml ($T_{stop} = 100$)	Rep1	-5934.27
	Rep2	-5941.21
	Rep3	-5928.63
	Rep4	-5930.18
	Rep5	-5940.13
GATSml ($T_{stop} = 200$)	Rep1	-5929.19
	Rep2	-5929.19
	Rep3	-5944.04
	Rep4	-5941.31
	Rep5	-5928.63
dnaml (default)		-5969.10
dnaml (jumble)		-5934.27

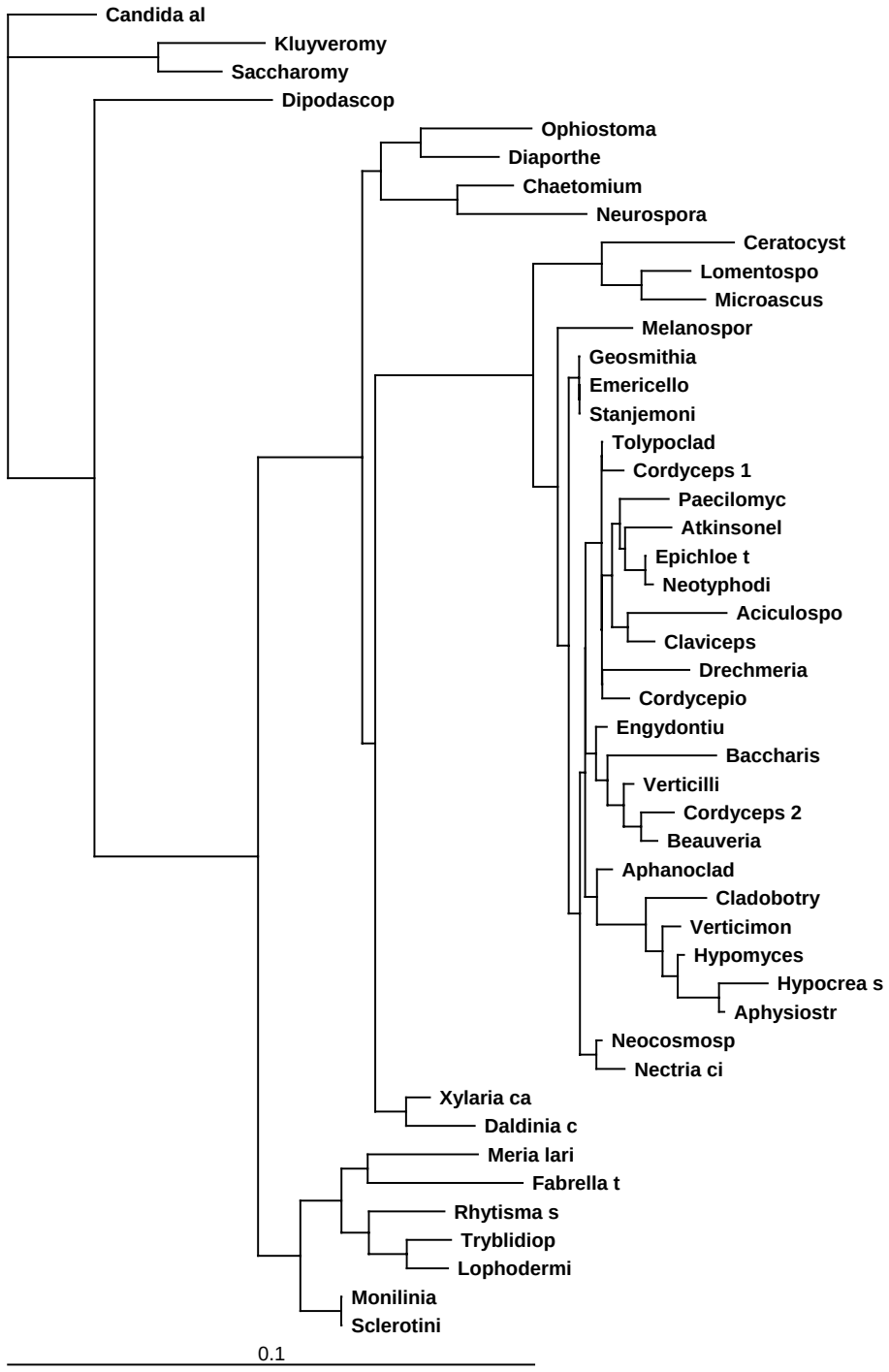


Figure 8.3: The maximum-likelihood tree produced by the GATSml program for the 47-taxon data set.

Table 8.2: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 10.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-2420.77	-2475.65	-2527.09	-2613.64	-2312.49	-2695.71	-2735.3	-2562.74	-2970.56	-3105.73
	Rep2	-2420.77	-2475.65	-2527.09	-2613.64	-2312.49	-2695.71	-2735.3	-2562.74	-2970.56	-3105.73
	Rep3	-2420.77	-2475.65	-2527.09	-2613.64	-2315.11	-2695.71	-2735.3	-2562.74	-2970.56	-3105.73
	Rep4	-2420.77	-2475.65	-2527.09	-2613.64	-2312.49	-2695.71	-2735.3	-2562.74	-2970.56	-3105.73
	Rep5	-2420.77	-2475.65	-2527.09	-2613.64	-2312.49	-2695.71	-2735.3	-2562.74	-2970.56	-3105.73
dnaml		-2418.58	-2475.65	-2527.09	-2615.09	-2312.02	-2696.94	-2735.3	-2568.62	-2972.56	-3105.73

Table 8.3: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 11.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-2495.26	-2619.91	-2991.79	-2567.38	-2999.08	-3125.71	-2841.3	-2678.9	-2796.2	-2871.29
	Rep2	-2495.26	-2619.91	-2991.79	-2567.38	-2999.08	-3125.71	-2841.3	-2678.9	-2796.2	-2871.29
	Rep3	-2495.26	-2619.91	-2991.79	-2567.38	-2999.08	-3125.71	-2841.3	-2678.9	-2796.2	-2871.29
	Rep4	-2495.26	-2619.91	-2991.79	-2567.38	-2999.08	-3125.71	-2841.3	-2678.9	-2796.2	-2871.29
	Rep5	-2495.26	-2619.91	-2991.79	-2567.38	-2999.08	-3125.71	-2841.3	-2678.9	-2796.2	-2871.29
dnaml		-2500.17	-2512.53	-2991.79	-2567.38	-2999.08	-3127.68	-2809.1	-2678.9	-2800.7	-2855.55

Table 8.4: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 12.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-3072.78	-2991.52	-3171.87	-2923.37	-2693.78	-3000.6	-2883.82	-2969.33	-3147.07	-2854.35
	Rep2	-3072.78	-2991.52	-3171.87	-2923.37	-2693.78	-3000.6	-2883.82	-2969.33	-3147.07	-2854.35
	Rep3	-3072.78	-2991.52	-3171.87	-2923.37	-2693.78	-3000.6	-2883.82	-2969.33	-3147.07	-2854.35
	Rep4	-3072.78	-2991.52	-3171.87	-2923.37	-2693.78	-3000.6	-2883.82	-2969.33	-3147.07	-2854.35
	Rep5	-3072.78	-2991.52	-3171.87	-2923.37	-2693.78	-3000.6	-2883.82	-2969.33	-3147.07	-2854.35
dnaml		-3072.78	-2991.52	-3173.84	-2888.86	-2703.9	-3000.6	-2874.76	-2969.33	-3147.14	-2854.42

Table 8.5: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 13.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-3402.61	-3016.19	-2722.25	-3110.31	-3204.68	-3047.96	-3084.71	-2498.4	-2798.82	-3743.26
	Rep2	-3402.61	-3016.19	-2722.25	-3110.31	-3204.68	-3047.96	-3084.71	-2498.4	-2798.82	-3743.26
	Rep3	-3402.61	-3016.19	-2722.25	-3110.31	-3204.68	-3047.96	-3084.71	-2498.4	-2798.82	-3743.26
	Rep4	-3402.61	-3016.19	-2722.25	-3110.31	-3204.68	-3047.96	-3084.71	-2498.4	-2798.82	-3743.26
	Rep5	-3402.61	-3016.19	-2722.25	-3110.31	-3204.68	-3047.96	-3084.71	-2498.4	-2798.82	-3743.26
dnaml		-3404.58	-2981.98	-2759.38	-3091.19	-3201.94	-3048.92	-3084.71	-2481.78	-2801.11	-3743.26

Table 8.6: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 14.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-2829.61	-3266.89	-3119.45	-3228.56	-2768.54	-2987.29	-3783.58	-3047.49	-3312.42	-3018.41
	Rep2	-2829.61	-3266.89	-3118.94	-3228.56	-2768.54	-2987.29	-3783.58	-3047.44	-3312.42	-3018.41
	Rep3	-2829.61	-3266.89	-3119.45	-3228.56	-2768.54	-2987.78	-3783.58	-3047.44	-3312.42	-3018.41
	Rep4	-2829.61	-3266.89	-3119.51	-3228.56	-2768.54	-2987.29	-3783.58	-3047.44	-3312.42	-3018.41
	Rep5	-2829.61	-3266.89	-3118.94	-3228.56	-2768.54	-2987.29	-3783.58	-3047.44	-3312.42	-3018.41
dnaml		-2859.16	-3251.60	-3118.98	-3236.45	-2715.56	-2989.90	-3783.58	-3018.11	-3312.45	-3019.98

Table 8.7: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 15.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-3293.19	-3157.53	-3247.47	-2680.23	-3388.51	-3325.83	-3408.39	-3132.82	-3353.13	-3109.21
	Rep2	-3293.19	-3157.53	-3247.47	-2680.23	-3388.51	-3325.83	-3408.39	-3132.82	-3353.13	-3109.21
	Rep3	-3293.19	-3157.53	-3247.47	-2680.23	-3388.51	-3325.83	-3408.39	-3132.82	-3353.13	-3109.21
	Rep4	-3293.19	-3157.53	-3247.47	-2680.23	-3388.51	-3325.83	-3408.39	-3132.82	-3353.02	-3109.21
	Rep5	-3293.19	-3157.53	-3247.47	-2680.23	-3388.51	-3340.56	-3408.99	-3132.82	-3353.13	-3109.21
dnaml		-3237.64	-3158.36	-3259.79	-2663.69	-3388.57	-3309.56	-3372.95	-3132.87	-3325.19	-3087.75

Table 8.8: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 16.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-3403.06	-2977.18	-3589.23	-3249.36	-3533.95	-3252.44	-3469.14	-3076.87	-3133.06	-3468.7
	Rep2	-3403.06	-2977.18	-3589.23	-3249.27	-3533.95	-3252.44	-3469.14	-3067.62	-3133.06	-3468.7
	Rep3	-3403.06	-2977.18	-3589.23	-3249.27	-3533.95	-3252.44	-3469.14	-3067.62	-3133.06	-3468.7
	Rep4	-3403.06	-2977.18	-3589.23	-3249.27	-3533.95	-3252.44	-3469.14	-3067.62	-3133.06	-3468.7
	Rep5	-3403.06	-2977.18	-3589.23	-3249.27	-3533.95	-3252.44	-3469.14	-3067.62	-3133.06	-3468.7
dnaml		-3415.20	-2926.48	-3607.07	-3217.86	-3539.74	-3253.33	-3455.09	-3039.03	-3090.48	-3480.5

Table 8.9: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 17.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-3894.74	-3268.89	-3659.29	-3519.86	-3567.09	-2987.94	-3538.02	-3344.01	-3280.69	-3319.91
	Rep2	-3894.74	-3268.89	-3659.29	-3519.86	-3567.09	-2987.94	-3538.02	-3344.01	-3263.07	-3319.91
	Rep3	-3894.74	-3268.89	-3659.29	-3519.86	-3576.38	-2987.95	-3538.02	-3344.01	-3263.07	-3319.91
	Rep4	-3894.74	-3268.89	-3659.29	-3519.86	-3567.09	-2987.94	-3538.02	-3344.01	-3263.07	-3319.91
	Rep5	-3894.74	-3268.89	-3659.29	-3519.86	-3567.09	-2987.95	-3538.02	-3344.01	-3263.07	-3319.91
dnaml		-3894.79	-3253.61	-3665.96	-3528.1	-3559.71	-3006.07	-3488.48	-3321.92	-3263.07	-3323.03

Table 8.10: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 18.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-3624.05	-3705.45	-3642.99	-3630.84	-3169.62	-3514.44	-3492.38	-3538.14	-3342.76	-3721.89
	Rep2	-3624.05	-3705.45	-3642.99	-3630.84	-3169.62	-3514.44	-3492.38	-3538.14	-3342.76	-3721.89
	Rep3	-3624.05	-3705.45	-3642.99	-3630.84	-3169.62	-3514.44	-3492.38	-3538.14	-3342.76	-3721.89
	Rep4	-3624.05	-3705.45	-3580.54	-3630.84	-3169.62	-3514.44	-3492.38	-3538.14	-3342.76	-3721.89
	Rep5	-3623.07	-3705.45	-3642.99	-3630.84	-3169.62	-3514.44	-3487.86	-3538.14	-3342.76	-3721.89
dnaml		-3590.53	-3713.33	-3583.02	-3619.59	-3194.58	-3511.85	-3457.68	-3546.53	-3289.57	-3722

Table 8.11: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 19.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-3780.01	-3727.91	-3196.28	-3768.28	-3729.8	-3654.79	-3656.8	-3620.78	-3517.55	-3199.84
	Rep2	-3780.01	-3729.49	-3196.28	-3768.28	-3729.8	-3654.79	-3656.8	-3620.78	-3518.87	-3199.84
	Rep3	-3780.01	-3729.49	-3196.28	-3766.67	-3724.74	-3654.79	-3656.8	-3620.78	-3518.87	-3199.84
	Rep4	-3780.01	-3727.91	-3196.28	-3766.67	-3729.8	-3654.79	-3656.8	-3620.78	-3518.87	-3199.84
	Rep5	-3780.01	-3727.91	-3196.28	-3768.28	-3729.8	-3654.79	-3656.8	-3620.78	-3518.87	-3199.84
dnaml		-3752.69	-3698.98	-3210.47	-3760.2	-3686.7	-3667.62	-3656.8	-3672.99	-3525.77	-3205.58

Table 8.12: The likelihood values derived from GATSml and dnaml on data sets with number of taxa = 20.

Program	Replication	Data1	Data2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	Data 10
GATSml	Rep1	-3400.46	-3620.8	-3720.42	-3669.3	-3615.14	-4047.64	-3499.44	-3342.41	-3627.22	-3895.75
	Rep2	-3400.46	-3620.8	-3720.42	-3669.3	-3615.25	-4047.64	-3499.44	-3342.41	-3627.22	-3896.34
	Rep3	-3400.46	-3620.8	-3720.42	-3669.3	-3615.25	-4047.64	-3495.02	-3342.41	-3627.22	-3896.17
	Rep4	-3400.46	-3620.8	-3720.42	-3669.3	-3615.25	-4047.64	-3495.02	-3342.41	-3627.22	-3895.93
	Rep5	-3400.46	-3620.8	-3720.42	-3669.3	-3615.14	-4047.64	-3499.44	-3342.41	-3627.22	-3895.81
dnaml		-3376.14	-3619.59	-3720.42	-3637.26	-3617.85	-4002.39	-3499.44	-3350.76	-3684.59	-3900.51

8.4 Discussion

The experimental results demonstrated in the previous section indicate that the proposed hybrid method is capable of achieving better maximum-likelihood trees comparing to the results from the dnaml program. On small-size data sets, the GATSml program performs slightly better than the default-setting dnaml program in average. On the 47-taxon data set, the GATSml program outperforms the default-setting dnaml program in all testing instances and has equally good performance as the dnaml program with the selected jumble option.

The major problem with the GATSml program is that it takes too much computation time for finding the maximum-likelihood trees. There are several reasons causing the slow computation. For example, the evaluation procedure of likelihood value is not efficient and the branch length optimization procedure converges slowly in some cases. Further investigation for improving the computation efficiency will be given in the next chapter.

Chapter 9

Summary and Future Research Directions

9.1 Summary

In Chapter 1, we introduce the basic concept of the phylogeny inference and four widely-used tree-building techniques: distance-based methods, maximum-parsimony methods, maximum-likelihood methods, and Bayesian methods. We focus on the maximum-parsimony and maximum-likelihood phylogeny inference. Their problem formulations are given in Chapter 2. In Chapter 2, we introduce the basics of tree topologies and several commonly-used methods for constructing a tree and rearranging a tree topology. A review of known topology searching methods for finding the maximum-parsimony and maximum-likelihood phylogenetic trees is given in this Chapter. The majority of the topology searching methods is using heuristic searches such as hill-climbing methods, simulated annealing and genetic algorithms.

In Chapter 3, we present a tabu search algorithm for the maximum-parsimony phylogeny inference. A two-array structure is used to represent each tree topology. We present two methods, *profile change* and *leaf swapping*, for the tree topology rearrangement. The performance analysis of the proposed tabu search algorithm is given in Chapter 4. We create a C++ program, *tabupars*, to perform this analysis. From the experimental results, we show that the tabupars program can achieve the globally optimal maximum-parsimony trees faster than the branch-and-bound-based dnapenny program.

In Chapter 5, a hybrid method which combines genetic algorithm and tabu search is presented for the maximum-parsimony phylogeny inference. Each tree topology is converted to its canonical form and the Newick format is used to represent the tree. The modified *subtree pruning and regrafting* method and the *nearest neighbor interchange* method are served as the crossover operator and mutation operator, respectively. During the searching process, a population is divided into *elite group* and *non-elite group*. A large proportion of the offspring is created by performing the crossover and mutation on the members in the elite group. The trees which have been recently selected into the elite group are prohibited from being selected again for a period of time. The performance of the hybrid method is investigated in Chapter 6. Two C++ programs, *GATSpars* and *GATSpars**, are created to implement the hybrid method for the maximum-parsimony phylogeny inference. A GA-based program, *GApars*, is also created for the comparison purpose. The experimental results show that the GATSpars program achieves the globally optimal maximum-parsimony trees with much fewer number of parsimony evaluations than the tabupars program. The GATSpars* program, which employs the sequential-addition technique to generate starting trees, although does not perform as good as PAUP*, shows better performance than the dnapars and GApars programs.

In Chapter 7, the hybrid method is modified for the maximum-likelihood phylogeny inference. The tree representation is revised to incorporate the branch lengths. The likelihood of each tree is evaluated based on the F84 model of evolution. We use a dynamic programming procedure and the Newton-Raphson method to optimize each branch length of a tree sequentially. In order to avoid spending too much time on evaluating the likelihood and optimizing the branch lengths of a non-optimal tree topology, a threshold is set to filter out the tree topologies with bad parsimony scores. The performance of the hybrid method for the maximum-likelihood phylogeny inference is presented in Chapter 8. We create a C++ program, *GATSml*, to implement this hybrid method. On the randomly generated data sets, the GATSml program shows slightly better performance than the default-setting dnaml program. On the 47-taxon data set, the GATSml program outperforms the default-setting dnaml program and has equally good performance as the dnaml program with selected jumble option.

9.2 Future Research Directions

Several suggestions for the future research are presented as follows.

1. The evaluation procedure for the parsimony scores and likelihood values needs to be improved. Studies have shown that the evaluation of a new tree produced by a topology rearrangement method, such as SPR and NNI, can be completed by simply evaluating a partial tree [28, 63]. This will greatly improve the efficiency of the searching process.

2. The branch length optimization procedure needs improvement. When a new tree is produced by a topology rearrangement method, usually not all the branch lengths need to be re-optimized [55, 63]. If the conditional likelihood of a subtree is stored in the corresponding interior node, then we are able to perform the branch length optimization only on the branches near the interior nodes where the topology rearrangement occurs.

3. In the current design of the proposed tabu search algorithm, we tabu the trees that have been selected in recent iterations. Whelan proposed a tabu search algorithm for the maximum-likelihood phylogeny inferences in 2007 [59], which will tabu the trees contain similar topologies. The tabu mechanism was shown to improve the convergency of the searching process.

4. As mentioned in Section 2.4, utilizing the information provided from the sequence data, such as re-weighting the sequence characters or partitioning the sequences according to their similarity, usually improves the convergence of the searching process [31, 45, 48]. The sequence information can also be incorporated in the tabu mechanism. For example, if two sequences are known to be distantly related, a tree which contains a clade of these two sequences should be tabued.

Bibliography

[1] D. Barker. LVB: parsimony and simulated annealing in the search for phylogenetic trees. *Bioinformatics*, 20:274–275, 2004.

[2] N. A. Barricelli. Esempi numerici di processi di evoluzione. *Methodos*, pages 45–68, 1954.

[3] J. Bł ażewicz, P. Formanowicz, F. Glover, M. Kasprzak, and J. Węglarz. An improved tabu search algorithm for dna sequencing with errors. In *Proceedings of the Metaheuristics International Conference, Angra dos Reis*, pages 69–75, 1999.

[4] H. J. Bremermann. Optimization through evolution and recombination. In M. C. Yovits, G. T. Jacobi, and G. D. Goldstein, editors, *Self-Organizing Systems*. Spartan Books, Washington D. C., 1962.

[5] D. G. Brown and I. M. Harrower. Integer programming approaches to haplotype inference by pure parsimony. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 3:141–154, 2006.

[6] L. L. Cavalli-Sforza and A. W. F. Edwards. Phylogenetic analysis: Models and estimation procedures. *American Journal of Human Genetics*, 19:233–257, 1967.

[7] A. Cayley. On the theory of the analytical forms called trees. *Philos. Mag.*, 13:19–30, 1857.

[8] M. A. Charleston. Hitch-hiking: A parallel heuristic search strategy applied to the phylogeny problem. *Journal of Computational Biology*, 8:79–91, 2001.

[9] C. B. Congdon. Gaphyl: A genetic algorithms approach to cladistics. In L. DeRaedt and A. Siebes, editors, *Lecture Notes in Computer Science*, number 2168, chapter

Principles of Data Mining and Knowledge Discovery, pages 67–78. Springer-Verlag , Berlin, 2001.

[10] C. Cotta. Scatter search with path relinking for phylogenetic inference. *European Journal of Operational Research*, 169:520–532, 2004.

[11] W. H. E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49:461–467, 1986.

[12] A. Dress and M. Krüger. Parsimonious phylogenetic trees in metric spaces and simulated annealing. *Advances in Applied Mathematics*, 8:8–37, 1987.

[13] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, editors. *Biological Sequence Analysis*. Cambridge University Press, New York, 1998.

[14] A. W. F. Edwards and L. L. Cavalli-Sforza. The reconstruction of evolution. *Annals of Human Genetics*, 27:105–106, 1963.

[15] A. W. F. Edwards and L. L. Cavalli-Sforza. *Phenetic and Phylogenetic Classification: Reconstruction of Evolutionary Trees*. Systematics Association Publ. No. 6, 1964.

[16] J. Felsenstein. Evolutionary trees from DNA sequences: A maximum likelihood approach. *Journal of Molecular Evolution*, 17:368–376, 1981.

[17] J. Felsenstein. *Inferring Phylogenies*. Sinauer Associates, Boston, MA, 2004.

[18] J. Felsenstein. *PHYLIP (Phylogeny Inference Package) Version 3.6*. Distributed by the author. Department of Genome Sciences, University of Washington, Seattle, 2005.

[19] W. M. Fitch. Toward defining the course of evolution: Minimum change for a specified tree topology. *Systematic Zoology*, 20:406–416, 1971.

[20] W. M. Fitch. On the problem of discovering the most parsimonious tree. *American Naturalist*, 111:223–255, 1977.

[21] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155:279–284, 1967.

[22] A. Fraser. Simulation of genetic systems by automatic digital computers. I. Introduction. *Aust. J. Biol. Sci.*, 10:484–491, 1957.

[23] M. Gen and R. Cheng. *Genetic Algorithms and Engineering Optimization.* Wiley interscience, New York, 2000.

[24] D. S. Gernandt and J. K. Stone. Phylogenetic analysis of nuclear ribosomal DNA places the nematode parasite, Drechmeria coniospora, in Clavicipitaceae. *Mycologia*, 91:993–1000, 1999.

[25] F. Glover. Tabu search - Part I. *ORSA Journal on Computing*, 1:190–206, 1989.

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

[27] R. L. Graham and L. R. Foulds. Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Mathematical Biosciences*, 60:133–142, 1982.

[28] S. Guindon and O. Gascuel. A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic Biology*, 52:696–704, 2003.

[29] D. Gusfield. Haplotyping by pure parsimony. In R. Baeza-Yates, E. Chavez, and M. Chrochemore, editors, *14th Annual Symposium on Combinatorial Pattern Matching (CPM 2003)*, volume 2676, pages 144–155. Springer LNCS, 2003.

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

[31] B. R. Holland, K. T. Huber, D. Penny, and V. Moulton. The minmax squeeze: Guaranteeing a minimal tree for population data. *Molecular Biology and Evolution*, 22:235–242, 2005.

[32] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press: Ann Arbor, MI, 1975.

[33] T. Jukes and C. Cantor. *Evolution of protein molecules. Mammalian Protein Metabolism*, volume 3. Academic Press, New York, 1969.

[34] K. Katoh, K. Kuma, and T. Miyata. Genetic algorithm-based maximum-likelihood analysis for molecular phylogeny. *Journal of Molecular Evolution*, 53:477–484, 2001.

[35] B. Larget and D. Simon. Markov chain Monte Carlo algorithms for the bayesian analysis of phylogenetic trees. *Molecular Biology and Evolution*, 16:750–759, 1990.

[36] A. R. Lemmon and M. C. Milinkovitch. The metapopulation genetic algorithm : An efficient solution for the problem of large phylogeny estimation. *Proceedings of the National Academy of Sciences*, 99:10516–10521, 2002.

[37] P. O. Lewis. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular Biology and Evolution*, 15:277–283, 1998.

[38] Y.-M. Lin, S.-C. Fang, and J. L. Thorne. A tabu search algorithm for maximum parsimony phylogeny inference. *European Journal of Operational Research*, 176:1908–1917, 2007.

[39] M. Lundy. Applications of the annealing algorithm to combinatorial problems in statistics. *Biometrika*, 72:191–198, 1985.

[40] H. Matsuda. Protein phylogenetic inference using maximum likelihood with a genetic algorithm. In *Pacific Symposium on Biocomputing, 1996, pp. 512-523*, 1996.

[41] B. Mau, M. A. Newton, and B. Larget. Bayesian phylogenetic inference via Markov chain Monte Carlo methods. *Biometrics*, 55:1–12, 1999.

[42] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–1092, 1953.

[43] B. Q. Minh, L. S. Vinh, A. von Haeseler, and H. A. Schmidt. pIQPNNI: Parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, 21:3794–3796, 2005.

[44] A. Moilanen. Searching for most parsimonious trees with simulated evolutionary optimization. *Cladistics*, 15:39–50, 1999.

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

[46] G. Olsen, H. Matsuda, R. Hagstrom, and R. Overbeek. FastDNAml: A tool for construction of phylogenetic trees of DNA sequences using maximum likelihood. *Comput. Appl. Biosci.*, 10:41–48, 1994.

[47] P. W. Purdom, P. G. Bradford Jr., K. Tamura, and S. Kumar. Single column discrepancy and dynamic max-mini optimizations for quickly finding the most parsimonious evolutionary trees. *Bioinformatics*, 16:140–151, 2000.

[48] D. L. J. Quicke, J. Taylor, and A. Purvis. Changing the landscape: A new strategy for estimating large phylogenies. *Systematic Biology*, 50:60–66, 2001.

[49] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstruction phylogenetic trees. *Molecular Biology and Evolution*, 4:406–425, 1987.

[50] L. Salter and P. D. K. Stochastic search strategy for estimation of maximum likelihood phylogenetic trees. *Systematic Biology*, 50:7–17, 2001.

[51] D. Sankoff. Minimal mutation trees of sequences. *SIAM Journal of Applied Mathematics*, 28:35–42, 1975.

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

[53] S. Sridhar, F. Lam, G. Blelloch, R. Ravi, and R. Schwartz. Efficiently finding the most parsimonious phylogenetic tree via linear programming. In *Proceedings of the 2007 International Symposium on Bioinformatics Research and Applications*, 2007.

[54] A. Stamatakis. An efficient program for phylogenetic inference using simulated annealing. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005.

[55] A. Stamatakis, T. Ludwig, and M. H. RAxML-III: A fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21:456C463, 2005.

[56] K. Strimmer and A. von Haeseler. Quartet puzzling: A quartet maximum-likelihood method for reconstructing tree topologies. *Molecular Biology and Evolution*, 13:964–969, 1996.

[57] D. L. Swofford. *PAUP*. Phylogenetic Analysis Using Parsimony (*and Other Methods). Version 4.* Sinauer Associates, Sunderland, Massachusetts, 2003.

[58] L. S. Vinh and A. von Haeseler. IQPNNI: Moving fast through tree space and stopping in time. *Molecular Biology and Evolution*, 21:1565C1571, 2004.

[59] S. Whelan. New approaches to phylogenetic tree search and their application to large numbers of protein alignments. *Systematic Biology*, 56:727–740, 2007.

[60] Z. Yang. Maximum likelihood estimation on large phylogenies and analysis of adaptive evolution in human influenza virus A. *Journal of Molecular Evolution*, 51:423–432, 2000.

[61] Z. Yang. PAML 4: a program package for phylogenetic analysis by maximum likelihood. *Molecular Biology and Evolution*, 24:1586–1591, 2007.

[62] M. Zachariasen and M. Dam. *Meta-heuristics: Theory and Applications*, pages 571–587. Kluwer Academic Publishers, Boston, 1996.

[63] D. J. Zwickl. *Genetic Algorithm Approaches for the Phylogenetic Analysis of Large Biological Sequence Datasets under the Maximum Likelihood Criterion.* PhD thesis, The University of Texas at Austin, 2006.