FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

An Efficient Algorithm for Phylogeny Reconstruction by Maximum Likelihood

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Information & Knowledge Management

eingereicht von

Lam-Tung Nguyen

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Priv.-Doz. Dr. Nysret Musliu
Univ.-Prof.Dr.Arndt von Haeseler

Wien, 14.04.2011

(Unterschrift Verfasser/in) (Unterschrift Betreuer/in)

Declaration

Lam-Tung Nguyen
Gymnasiumstr. 85/459
1190 Wien

"Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit einschlielich Tabellen, Karten und Abbildungen, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, 14.04.2011

Lam-Tung Nguyen

An Efficient Algorithm for Phylogeny Reconstruction by Maximum Likelihood

Abstract

Understanding the evolutionary relationships among species has been of tremendous interest since Darwin published the Origin of Species (Darwin, 1859). The evolutionary history (phylogeny) of species is typically represented as a phylogenetic tree. Nowadays, the reconstructing of the evolutionary history is still a major research topic. With the rise of molecular sequencing technologies, advanced computational approaches have been proposed to reconstruct phylogenies. Maximum likelihood is a statistical method for reconstructing phylogeny which gives better estimate of the true tree than those produced by other approaches. Howerver, maximum likelihood, is highly computational expensive.

Therefore, different heuristics haven been proposed to solve this NP-hard problem. Among these, IQPNNI (Vinh and von Haeseler, 2004) has been shown to have highly regarded results. Nevertheless, it requires a lot of computation time. In the present work we introduce an improved version of the IQPNNI algorithm called IQ-Tree. Here we focus on improving the runtime performance of IQPNNI by proposing a fast and reliable search algorithm for phylogeny reconstruction.

To this end, we used the metaheuristic Iterated Local Search (ILS) as our underlying search framework, from which the possibility of using the search history to improve performance has been identified. Based on the search history we propose a strategy that directs the search to move quickly through regions, in which better trees are unlikely to be found. Furthermore, we introduce a vectorization technique to speed up computations in the tree evaluation function. Our results showed that IQ-Tree runs two to four times faster than IQPNNI, while the reconstructed phylogenetic trees have equal or better likelihood than those produced by IQPNNI.

Acknowledgements

I want to thank my supervisor Priv.-Doz. Dr. Nysret Musliu for the sincere encouragement and support he gave me during the course of my work. His interesting lectures on artificial intelligence and machine learning have provided great inspiration for me to start working on the thesis.

I want to express my gratitude to Univ.-Prof.Dr.Arndt von Haeseler for giving me the chance to work on the project. His guidance and advice were invaluable to me.

I would like to pay special tribute to Dr.Bui Quang Minh for always being my great teacher whenever I needed. Without his assistance I would not have achieved what I set out to do. These few lines of words probably won't be enough to show my appreciation of all his support.

I also thank Dr.Heiko Schmidt for the fruitful discussions in our bi-weekly group meetings. Working with all the other colleagues at the Center of Integrative Bioinformatics Vienna is always my great pleasure.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this chapter we explain the phylogeny reconstruction problem. Then, we discuss some methodologies and state of the art techniques for solving the problem. Finally, we summarize our main contributions to the problem and outline the structure of the remaining part of the thesis.

1.1 Motivation

The foundation of evolutionary biology was laid by Charles Darwin in his famous work "On the Origin of Species" (Darwin, 1859). Darwin proposed that life on Earth has evolved from a common ancestor. "Nothing in biology makes sense except in the light of evolution", wrote the prominent geneticist and evolutionary biologist Theodosius Dobzhansky (Dobzhansky, 1973).

During the evolution of species, traits are inherited and mutations are accumulated. This leads to divergence in the evolutionary paths. Based on common characteristics of contemporary species, the evolutionary relationships between them can be reconstructed and represented in a so-called phylogeny or phylogenetic tree. Figure 1.1 illustrates a phylogenetic tree of the great apes species, including human. Studying evolution helps us understand biological phenomena. Typical examples of this can be taken from the field of medicine. In order to prevent pathogenic diseases, researchers must understand the evolutionary pattern of disease-causing organisms. Hereditary diseases can be controlled by studying evolutionary histories

of the disease-causing genes. The scientific field studying evolutionary relationships among various groups of organisms is called *phylogenetics* (Berkeley, 2010b).

Reconstructing phylogeny is the central topic in phylogenetics because the shape of phylogenetic trees are often unknown or controversial in many cases. In order to reconstruct a phylogeny, we have to collect and analyze data about the characters of each organism of interest. Characters are heritable traits that can be compared across organisms, such as morphological characteristics, genetic sequences, and behavioral traits (Berkeley, 2010a).

Advances in sequencing technologies have led to the emergence of molecular evolution research in the 1960s. Phylogenetic tree reconstruction is done much more efficiently using molecular sequences (e.g. DNA, Protein). Because of its compactness and stable structure, genetic sequences have become the main data used for reconstructing phylogenetic trees (Zuckerkandl, 1987).

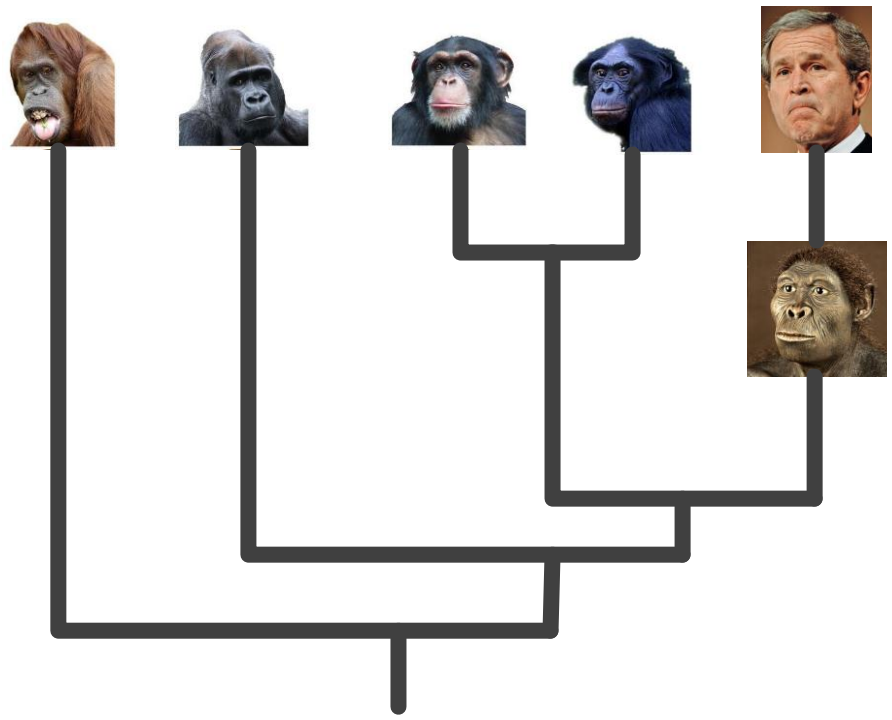

Figure 1.1: Phylogenetic tree of the great apes.

1.2 State of the art phylogeny reconstruction

Although the concept of phylogeny has been around for 150 years, statistical, computational, and algorithmic methods for the problem have only been applied for about 50 years (Felsenstein, 2004).

Reconstructing a phylogeny is an estimation problem. Genetic sequences are only available for contemporary species whereas those of their extinct ancestors are by and large missing. In order to estimate a phylogeny, one needs an objective function to evaluate different tree topologies and a search strategy to find the best tree, with respect to the objective function. Phylogenetic reconstruction methods are classified into four main categories on the basis of their objective functions: maximum parsimony, minimum evolution, maximum likelihood, and Bayesian methods (Barton, 2007). Since phylogeny reconstruction methods are only based on assumptions about different aspects of the evolution of the species, one cannot guarantee that the reconstructed phylogeny exactly reflects the true evolutionary history.

The starting tree for a search procedure is usually built by a fast tree reconstruction method, e.g. UPGMA (Sokal and Michener, 1958) or Neighbor-Joining (NJ) (Saitou and Nei, 1987). Such methods use pairwise distances between species to construct the tree by a greedy algorithm. Since the number of possible tree topologies grows exponentially with the number of species (Felsenstein, 1978), finding the optimal tree using exhaustive searches is NP-complete (Foulds and Graham, 1982; Day and Sankoff, 1986; Addario-Berry *et al.*, 2003).

Therefore, numerous heuristic methods have been introduced to tackle the tree search problem. The techniques used range from simple hill climbing algorithms to sophisticated global optimization strategies. In hill climbing algorithms, local improvements on the current tree are made by using tree re-arrangement techniques such as nearest neighbor interchange (NNI), subtree pruning and re-grafting (SPR), tree bisection and reconnection (TBR) (Felsenstein, 2004). The search procedure stops when no local improvements can be found. Hill climbing techniques have been shown to be very effective for the phylogeny reconstruction problem, and programs that implement such techniques are among the most widely used. Notable programs are Phylip (Felsenstein, 1995), PAUP* (Swofford, 2002), PhyML (Guindon and Gas-

cuel, 2003; Guindon *et al.*, 2010), RAxML (Stamatakis, 2006). Hill climbing, however, tends to end up in a local optimum and is not guaranteed to find the best tree (the global optimum) (Russell and Norvig, 2009). Efforts have therefore been made for the quest of finding the global optimal tree. Global optimization algorithms for finding phylogenetic tree are either ad-hoc techniques or based on well-established metaheuristics. Notable implementations of different metaheuristics for the problem are: MetaPIGA (Lemmon and Milinkovitch, 2002) and GARLI (Zwickl, 2006) (genetic algorithm), LEAPHY (Whelan, 2007) (tabu search). Vinh and von Haeseler (2004) also proposed an efficient global optimization strategy IQPNNI (see chapter 4).

1.3 Contributions

In this thesis we developed an efficient tree search algorithm for the problem of inferring a phylogeny based on the maximum likelihood principle. The idea of our algorithm is based on the work done in PhyML (Guindon and Gascuel, 2003) and IQPNNI (Vinh and von Haeseler, 2004). Our aim was to provide an algorithm that is able to reconstruct good phylogenies in short time.

Within this work we achieved the following contributions:

1. We showed that the tree search algorithm used by Vinh and von Haeseler (2004) can be formulated as the metaheuristic Iterated Local Search (ILS) (Lourenco *et al.*, 2003). Since our work is derived from IQPNNI, we decided to use ILS as the underlying principle to investigate the different components of the algorithm.

2. By studying ILS we identified the possibility of employing the search history to enhance performance. To this end, we introduced an efficient search heuristic called IQ-Tree which utilizes data collected from previous tree search attempts to adjust the thoroughness of searching the tree space. The heuristic helps the search to find trees of good quality quickly by eliminating search regions, in which a new better tree is unlikely to be found.

3. We employed parallelization technique in IQ-Tree to speed up the time-consuming likelihood computations. The advantage of this technique is that it can be used on conventional desktop computers with only one Central Processing Unit (CPU) and does not require specialized parallelization infrastructure.

4. Our results showed that IQ-Tree reconstructs trees with the same quality as those by IQPNNI while only needs a fraction of the time required by IQPNNI.

1.4 Thesis Structure

The remaining parts of the thesis are structured as follows:

- Chapter 2 gives a quick overview of the fundamental aspects in phylogeny reconstruction. Here the focus is given to the maximum likelihood method.

- Chapter 3 describes essential aspects of the metaheuristic Iterated Local Search that acts as the underlying framework for our tree search algorithm.

- Chapter 4 presents our core tree search framework based on the IQPNNI algorithm.

- Chapter 5 contains details of our performance optimization techniques. First, we present extensive analyses of the search behavior of the IQPNNI algorithm. Then, we introduce a new search heuristic which employs the search history to make the search more efficient. Second, we present a parallelization strategy to reduce the tree likelihood computation time. At the end we show the computational results of IQ-Tree in comparison with IQPNNI.

- Chapter 6 highlights our main contributions and give some aspects about the future work.

Chapter 2

Essentials of Phylogenetic Inference

This chapter gives a short introduction to the biological and computational aspects of molecular phylogeny reconstruction. It presents the problem of phylogenetic tree reconstruction while introducing the terminologies used throughout the thesis.

2.1 Genetic Information

Since the advent of molecular biology, it has been widely understood, that changes in genetic information are the driving force of evolution. The genetic information of a species is stored in the *genome* and encoded in *DNA* (deoxyribonucleic acid), or for some viruses in *RNA* (ribonucleic acid) (Lemey *et al.*, 2009). The *DNA* is the building block of genes which encode proteins.

Genetic data is usually represented by *nucleotide* or *amino acid* sequences. Nucleotides are molecules that make up the elementary units of DNA and RNA. Nucleotides exist in five different types: Cytosince, Guanine, Adenine, Thymine and *U*racil. A nucleotide sequence can thus be stored as sequences of characters of the alphabets C, G, A, [T—U] (Thymine is only presented in DNA whereas Uracil is found in RNA). Amino acids are the building blocks of protein that exist in 20 different biologically relevant types (Table 2.1). Protein sequences are generally represented by a string of one-letter amino acid abbreviations (Lemey *et al.*, 2009).

Name	3-letter abbr.	1-letter abbr.	Name	3-letter abbr.	1-letter abbr.
Alanine	Ala	A	Methionine	Met	M
Cysteine	Cys	C	Asparagine	Asn	N
Aspartic acid	Asp	D	Proline	Pro	P
Glutamic acid	Glu	E	Glutamine	Gln	Q
Phenylalanine	Phe	F	Arginine	Arg	R
Glycine	Gly	G	Serine	Ser	S
Histidine	His	H	Threonine	Thr	T
Isoleucine	Ile	I	Valine	Val	V
Lysine	Lys	K	Tryptophan	Trp	W
Leucine	Leu	L	Tyrosine	Tyr	Y

Table 2.1: Twenty amino-acids (Vandamme, 2003).

In recent years, advances in sequencing technologies have led to an exponential growth of genetic sequence data. There are public sequence databases containing millions of sequences, for example GenBank (Benson *et al.*, 2005) or EMBL (Kulikova *et al.*, 2006). To keep up with the fast-growing number of biological data, biological sequence analysis methods, including phylogenetic inference, need to be constantly improved with respect to computational efficiency.

2.2 Sequence Alignment

Throughout the evolutionary process, an organism may sometimes changes its genetic information. Those changes are called mutations. Mutations are usually caused by radiation, viruses, mutagenic chemicals as well as errors that occur during cell replication (Bertram, 2000). The accumulation of mutations is the evidence of evolutionary development. Sequences are called homologous if they have evolved from a common ancestral sequence (Durbin *et al.*, 2002). Given a set of homologous genetic sequences, one of the common tasks is to compare them to identify regions of similarity that may be results of functional, structural and evolutionary relationships

between the sequences (Mount, 2004). Therefore, sequence alignment is a frequently used technique for such purpose.

We now describe the method for aligning nucleotide sequences. For protein sequences the same approach can be applied. A change in a single nucleotide is called point mutation. Point mutation can be one of the following operations:

- Substitution: replace a nucleotide by another.
- Insertion: insert one nucleotide into the sequence.
- Deletion: delete one nucleotide from the sequence.

In aligned sequences, gaps are inserted in between so that homologous nucleotides are aligned in the same column. Figure 2.1 shows an example of aligning two DNA sequences (pairwise alignment). In descendant 1 one insertion event happened. In descendant 2 substitution and deletion events both occurred. Insertions and deletions are represented in the alignment by the gap characters ('-').

Sequences which are short and very similar can be aligned manually. However, most real-life sequences are very long and contain a large number of highly variable character patterns. Hence, alignments need to be done by computational approaches. Alignment algorithms usually use some scoring scheme, based on which the alignment with highest score is sought (Durbin *et al.*, 2002). A scoring scheme could be as simple as follows: the score of a match is +1, whereas -1 is for a mismatch and -2 is for an *indel* [1]. The pairwise alignment in Figure 2.1 has 12 matches, 1 mismatch and 2 indels, yielding a score of 7. In practice scoring schemes are much more sophisticated since they require careful consideration of the biological meanings.

Multiple sequence alignment (MSA) is an extension of pairwise alignment where more than two sequences are aligned. Figure 2.2 shows an example of MSA of Human, Chimpanzee, Gorilla, Rhesus. MSA acts as the input data for the reconstruction of phylogenetic trees.

[1] Since it is not possible to distinguish between insertion and deletion, they are usually referred to as *indels*.

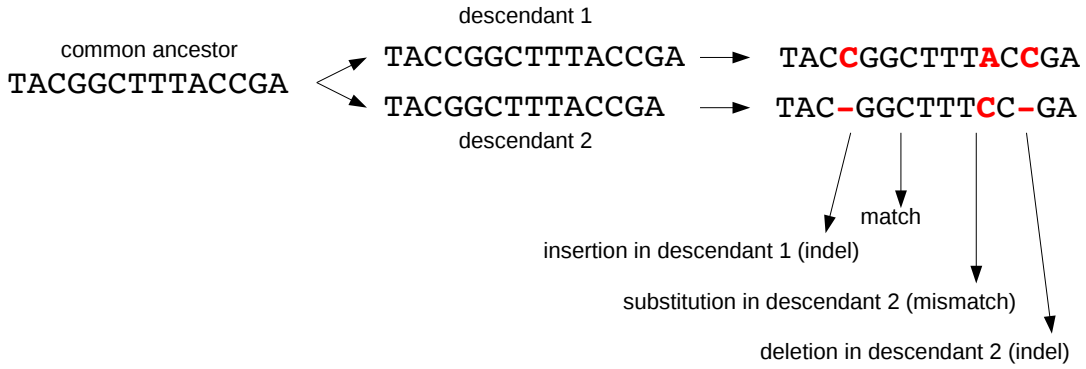

Figure 2.1: Pairwise alignment of DNA sequences.

While two sequences can be efficiently aligned using dynamic programming (Needleman and Wunsch, 1970; Smith and Waterman, 1981), aligning multipe sequences requires significantly more computational effort. Multiple sequence alignment was shown to be an NP-complete combinatorial optimization problem (Wang and Jiang, 1994; Elias, 2006). Various heuristic methods have therefore been developed for the task of efficiently aligning multiple sequences, such as CLUSTAL W (Thompson *et al.*, 1994), T-COFFEE (Notredame *et al.*, 2000), MUSCLE (Edgar, 2004).

Figure 2.2: A example of a multiple sequence alignment for human, chimpanzee, Gorilla and Rhesus. Columns in the alignment are called sites.

	1	2	3	4	5	6	7	8	9	10	...
Human	A	T	G	C	G	C	A	T	C	A	...
Chimpanzee	A	T	G	C	-	G	G	T	G	T	...
Gorilla	G	-	G	A	G	A	C	T	T	A	...
Rhesus	T	-	C	A	A	G	G	T	C	T	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

2.3 Phylogenetic Tree Reconstruction Methods

Once a multiple sequence alignment is provided, a phylogenetic tree can be constructed based on the alignment. Sequences are represented by leaves in the tree. As mentioned in 1.2, the phylogenetic methods are classified into four main categories: parsimony, minimum evolution, likelihood and Bayesian methods. The following short descriptions give a quick overview of each method (Barton, 2007):

- **Maximum parsimony**: The parsimony score of a tree is computed as the minimum number of character changes that is required to explain the variation observed in the alignment. Different tree topologies are compared according to their parsimony scores and the optimal tree is the one that has the lowest score (requiring the fewest changes).

- **Minimum evolution**: The evolutionary distances between all pairs of sequences are first estimated and stored in a distance matrix. The optimal tree is the tree, whose branch patterns and lengths best approximate the distance matrix.

- **Maximum likelihood**: Trees are evaluated with a model of evolution, which reflects the substitution probabilities of characters. The score of a particular tree is the likelihood of the observed characters having evolved according to the tree model. The optimal tree is the tree with the highest likelihood score.

- **Bayesian**: Bayesian inference can be used to construct a phylogenetic tree in a manner closely related to the maximum likelihood methods. But unlike the maximum likelihood methods, Bayesian methods try to maximize the posterior probability of the tree given the alignment. Bayesian methods generally employ Markov chain Monte Carlo (MCMC) methods to sample the distribution of trees based on a prior probability distribution.

Accurately reconstructing the phylogeny of species requires a method that is capable of incorporating multiple mutational events at the same site (Holder and Lewis, 2003). Maximum likelihood (ML) is such method. In the following we will describe the ML method for phylogenetic tree reconstruction.

2.4 Phylogenetics by Maximum Likelihood

In ML a statistical framework is used to infer phylogenetic tree. A hypothesis about the phylogeny is made and the hypothesis is judged by how well it predicts the observed data, represented by MSA. Trees are selected based on their likelihoods of "producing" the data. To calculate the tree likelihood one needs a model of sequence evolution that reflects the probability of the various mutational events.

2.4.1 Models of Sequence Evolution

Models of sequence evolution provide mechanism for calculating probabilities of changes/substitutions occurs while sequences evolve. These models are based on several assumptions about the substitution process of nucleotides or amino acids (Felsenstein, 1981, 2004):

- *Markov process*: The probability of character state i changing to character state j does not depend on the ancestral state prior to i.
- *Time-continuous*: Substitutions can happen at any time.
- *Homogeneous*: Substitution probabilities do not change in different parts of the tree.
- *Stationary*: The overall character frequencies π_i of the nucleotides or amino acids are at equilibrium and remain constant, where π_i is the frequency of character i.
- *Time-reversible*: Mutation in either direction is equally likely. The probability of character i mutating to character j is the same as the probability of the back mutation j to i within the same amount of time. For the computations based on the model it is thus unimportant to know which sequence is the ancestor and which is the descendant. Therefore, the likelihood of the tree generally does not depend on how the tree is rooted and likelihood computation can be done on unrooted tree.

Evolutionary models are often described by substitution rate matrices, which contain the rate (number of substitutions per unit of evolutionary time) at which a sequence character is replaced by the alternative. For DNA sequences the evolutionary model is represented as a 4×4 instantaneous rate matrix Q:

$$Q = \begin{pmatrix} -\mu(a\pi_C + b\pi_G + c\pi_T) & \mu a\pi_C & \mu b\pi_G & \mu c\pi_T \\ \mu a\pi_A & -\mu(a\pi_A + d\pi_G + e\pi_T) & \mu d\pi_G & \mu e\pi_T \\ \mu b\pi_A & \mu d\pi_C & -\mu(b\pi_A + d\pi_G + f\pi_T) & \mu f\pi_T \\ \mu c\pi_A & \mu e\pi_C & \mu f\pi_G & -\mu(c\pi_A + e\pi_G + f\pi_T) \end{pmatrix}$$

Matrix Q is the most general form of a time-reversible substitution model, which is also called the **G**eneral **T**ime-Reversible **M**odel (GTR) (Tavaré, 1986). The rows and columns of the matrix both correspond to the nucleotide bases A, C, G and T. μ represents the mean instantaneous substitution rate and a, b, c, d, e, f are the relative rate parameters. The product of the mean instantaneous and a relative rate parameter makes up the *rate parameter*. $\pi_A, \pi_C, \pi_G, \pi_T$ are the frequencies of the bases A, C, G and T, respectively. For example, the instantaneous rate of substitution from C to A is $\mu a\pi_A$. The diagonal elements of Q is chosen, so that the sum of elements in each row is equal zero. The GTR model has ten parameters including six substitution rates and four equilibrium base frequencies. Since the four frequency parameters sum up to one ($\pi_A + \pi_C + \pi_G + \pi_T = 1$) and time is measured in substitution ($\mu = 1$), the number of parameters is reduced to eight (Swofford *et al.*, 1996).

Almost all DNA substitution models proposed to date are special variants of the GTR model. If we assume that the equilibrium frequencies of all nucleotide bases are the same ($\pi_A = \pi_C = \pi_G = \pi_T = 0.25$) and all substitution types occur at the same rate ($a = b = c = d = e = f = 1$) then our substitution model becomes the JC69 model (Jukes and Cantor, 1969). Kimura (1980) introduced the model K2P with rate parameters for two types of substitution: transition ($A \longleftrightarrow G$ or $C \longleftrightarrow T$) and transversion ($A \longleftrightarrow T, G \longleftrightarrow T, A \longleftrightarrow C,$ and $C \longleftrightarrow T$). The model F81 with different base frequencies was introduced by Felsenstein (1981). Hasegawa *et al.* (1985) unified the two previous models into a four parameter model HKY85. Figure

2.3 shows the relationships between different DNA substitution models.

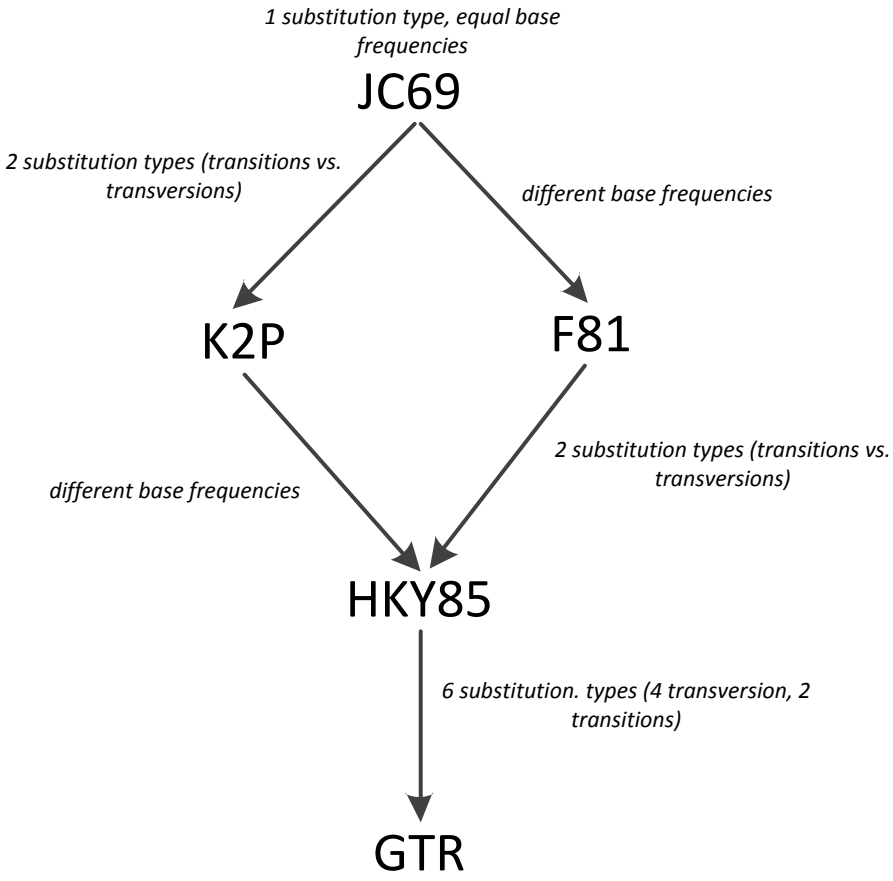

Figure 2.3: Substitution Models. Adapted from (Swofford *et al.*, 1996).

The instantaneous rate matrix Q only provides the substitution rate between pairs of nucleotides for an instant time dt. In order to calculate the likelihood, we need to derive the probability of change from one nucleotide to another nucleotide during a time period t. The matrix containing the substitution probabilities between pairs of nucleotides during time t is called the *transition matrix*. The elements of the transition matrix can be computed from the rate matrix via exponentiation:

$$P(t) = e^{Qt} \tag{2.1}$$

If Q is diagonalizable, the matrix exponential can be computed directly by decomposing Q into its eigenvalues and eigenvectors (Strimmer and von Haeseler, 2003). Let $Q = U\lambda U^{-1}$ be the diagonalization of Q, where λ is a matrix whose diagonal elements $\{\lambda_i\}$ are the eigenvalues of Q and the column vectors of U are the corresponding eigenvectors of Q then

$$P(t) = e^{Qt} = e^{U\lambda tU^{-1}} = Ue^{\lambda t}U^{-1} \tag{2.2}$$

since λ is a diagonal matrix:

$$e^{\lambda t} = \begin{pmatrix} e^{\lambda_1 t} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & e^{\lambda_4 t} \end{pmatrix} \tag{2.3}$$

In a similar way a protein substitution model is described by 20×20 rate matrices. Substitution models for protein can be divided into two types: *empirical* and *mechanistic* models (Yang, 2006). *Empirical* models are constructed by analyzing a large number of sequence data. *Mechanistic* models, in the contrary, take into account the biological mechanisms involved in the substitution process. While with DNA sequence data mechanistic models are used, the construction of amino acid replacement models are usually focused on the empirical approach. Some of the commonly used amino acid substitution models are: Dayhoff (Dayhoff and Schwartz, 1978), JTT (Jones *et al.*, 1992), WAG (Whelan and Goldman, 2001).

2.4.2 Computing Likelihood

In this section we will follow the notations from (Durbin *et al.*, 2002). Assuming that a model of sequence evolution has been specified, we now can compute the likelihoods of different trees. Given a set X of n sequences x_j for $j = 1, ..., n$. T is a tree with n leaves where leave j represents sequence x_j and E is the set of all branch lengths of the edges of T . $P(X|T, E)$ is then the likelihood of the data X given the tree topology T and its branch lengths E.

In order to compute the likelihood, two assumptions are made (Felsenstein, 2004):

1. Evolution in different lineages is independent.

2. Evolution in different sites is independent.

If x and y are two nodes of a branch with length t, where y represents the ancestral sequence of x, then $P(x|y,t)$ is the probability of y evolving to x after an evolutionary time t. Having made the first assumption, the probability of a phylogenetic tree is computed as the the product of all the probabilities $P(x_i|y_j,t_k)$, one for each branch.

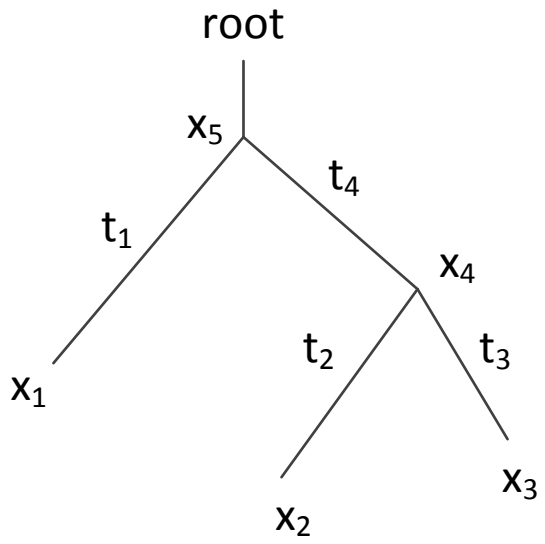

Figure 2.4: A example of a tree with three sequences and the associated branch lengths. Adapted from (Durbin *et al.*, 2002).

Figure 2.4 shows an example of a phylogenetic tree containing three contemporary sequences x_1, x_2 and x_3 (external nodes or leaves) with a set of specific ancestral sequences x_4, x_5 (internal nodes). The branch lengths of the tree are denoted by t_1, t_2, t_3 and t_4. The probability of the data given the tree in Figure 2.4 is calculated as follows:

$$P(x_1, ..., x_5|T, E) = P(x_5)P(x_1|x_5, t_1)P(x_4|x_5, t_4)P(x_2|x_4, t_2)P(x_3|x_4, t_3) \quad (2.4)$$

where $P(x_5)$ denotes the probability of the sequence x_5 being the root of the tree.

The second assumption about the evolution in different sites being independent allow us to compute the probability of each site separately and then multiply them

to obtain the final likelihood of the tree. Equation 2.4 can therefore be used to compute the probability of site S_i if we consider x_i to be a single character of the corresponding sequence at the respective site. Suppose that the sequence alignment has m sites, the likelihood of the tree is:

$$P(X|T,E) = \prod_{i=1}^{m} P(S_i|T,E) \tag{2.5}$$

Multiplying a large number probability values may cause numerical instability. To avoid this, the likelihood of a tree is computed as the sum of the logarithms of the likelihood values of all the sites.

From now on we will talk about likelihood computation in the context of a single site and x_i is referred to as a character of sequence i at the respective site. Since the ancestral sequences are unknown and in order to obtain the likelihood of the known contemporary sequences for a given tree topology, we need to sum over all the probabilities of the tree having all possible representation of the ancestral nodes. Equation 2.6 shows how the likelihood of the tree in Figure 2.4 can be computed for site S_i.

$$P(S_i|T,E) = \sum_{x_4 \in Z} \sum_{x_5 \in Z} P(x_1, x_2, x_3, x_4, x_5|T,E) \tag{2.6}$$

where Z is the set of possible character states, e.g. $Z = \{A, C, G, T\}$ for DNA. Combining Equation 2.4 and Equation 2.6 we then have:

$$P(S_i|T,E) = \sum_{x_4 \in Z} \sum_{x_5 \in Z} P(x_5)P(x_1|x_5,t_1)P(x_4|x_5,t_4)P(x_2|x_4,t_2)P(x_3|x_4,t_3) \tag{2.7}$$

Felsenstein's Pruning Algorithm

Since the number of terms in Equation 2.7 grows exponentially with the number of species, computing the likelihood using this method is intractable. Therefore, Felsenstein (1981) introduced a dynamic programming algorithm for rapidly computing the tree likelihood and thus speeding up the whole computation. The algorithm was called *pruning algorithm*.

The pruning algorithm is based on the possibility that one can move the summation signs in Equation 2.7 to the right and place them directly before the related

terms. We can rewrite Equation 2.7 as follows:

$$P(S_i|T,E) = \sum_{x_5 \in Z} P(x_5)P(x_1|x_5,t_1)\left(\sum_{x_4 \in Z} P(x_4|x_5)P(x_2|x_4,t_2)P(x_3|x_4,t_3)\right) \quad (2.8)$$

This suggests that the conditional likelihoods of different subtrees can be computed in a recursive manner. We call these conditional likelihoods *partial likelihoods*. The pruning algorithm computes the likelihood by walking up the tree from the leaves to the root, using post-order traversal (Durbin *et al.*, 2002).

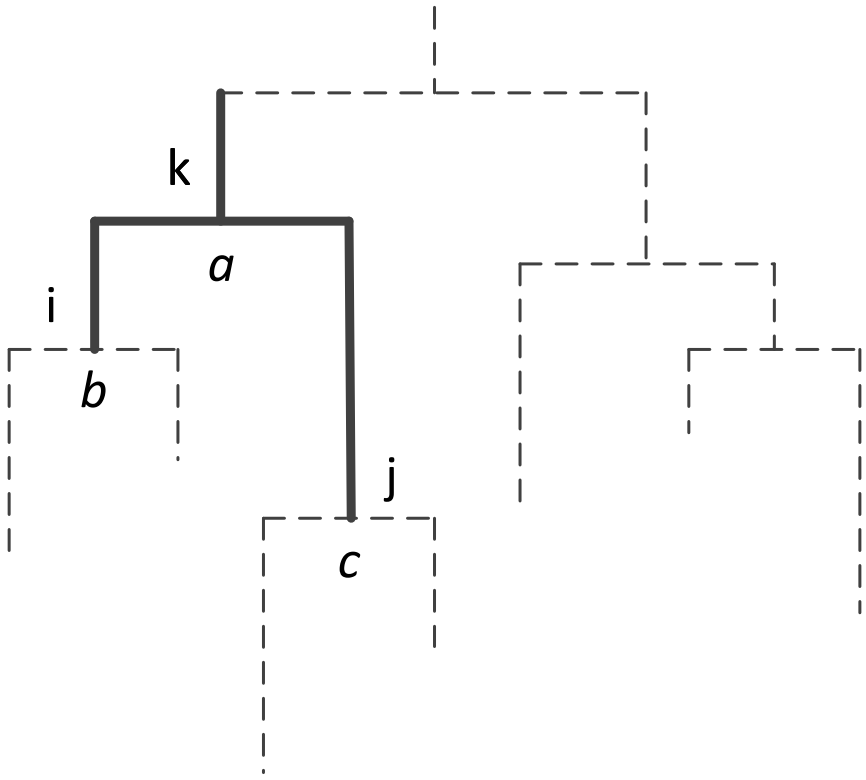

Figure 2.5: Computing partial likelihood on subtrees. Adapted from (Durbin *et al.*, 2002).

A phylogenetic tree can be either *bifurcating* or *multifurcating*. Each internal node of a bifurcating tree has exactly two descendants, whereas an internal node of a multifurcating tree can have more than two descendants. For the sake of simplicity we restrict the tree to be bifurcating. For a rooted bifurcating phylogenetic tree with n leave sequences there are $2n-1$ nodes. Given the character state at node k is a

($k = 1 \ldots 2n-1$, where node $2n-1$ is the root), let $P(L_k|a)$ denotes the partial likelihood of all the leaves below node k. Let $x_u^1 \ldots x_u^n$ denote the characters at site u of the n sequences $x^1 \ldots x^n$. $P(L_k|a)$ can be computed from the probabilities $P(L_i|b)$ and $P(L_j|c)$ for all possible b and c, where i and j are the descendant nodes of k (see Figure 2.4.2). The procedure of Felsenstein's pruning algorithm can be described as follows (Durbin *et al.*, 2002):

- Initialization:

  Set $k = 2n-1$

- Recursion: Compute $P(L_k|a)$ for all a.

  If k is a leaf node:

  Set $P(L_k|a) = 1$ if $a = x_u^k$, otherwise $P(L_k|a) = 0$.

  If k is not a leaf node:

  Compute $P(L_i|a)$, $P(L_j|a)$ for all a at the child nodes i,j.

  Set $P(L_k|a) = \sum_{b,c} P(b|a,t_i)P(L_i|b)P(c|a,t_j)P(L_j|c)$.

- Termination:

  The final likelihood of site $u = P(S_u|T,E) = \sum_a P(L_{2n-1}|a)\pi_a$.

2.4.3 Optimizing Parameters

Maximum likelihood estimation (MLE) is a popular statistical method for finding values of the model parameters that maximize the likelihood of the data. In phylogenetic context the model of MLE contains a tree (comprised of its topology and branch lengths) and a model of sequence evolution, where the data is the sequence alignment. The problem of finding the maximum likelihood phylogeny is thus a multidimensional optimization problem consisting of continuous and discrete optimization (Bryant D., 2005). In the course of searching for the optimal tree with respect to the likelihood score, the following optimizations are required:

1. Optimization of the parameters of the substitution model, including the relative rate parameters and sometimes the transition/transversion rate ratio, depending on the models (continuous optimization).

2. Optimization of the branch lengths of the tree (continuous optimization).

3. A search over all possible tree topologies (discrete optimization).

Each optimization step is usually done independently, while the other parameters are fixed. For optimizing model parameters, one can use different standard numerical optimization techniques, which simultaneously update all parameters. Methods like BFGS's method (Broyden Fletcher Goldfarb Shanno) (Gill *et al.*, 1981) and Brent's method (Brent, 2002) are the most commonly used. Optimization of the branch lengths can typically be done by an iterative approach, in which each branch is optimized separately by Newton's (Press *et al.*, 2007) or Brent's method. For a given tree, one could apply multidimensional optimization for both substitution parameters and branch lengths using Newton's method. However, this approach is difficult to implement and the computation can be quite slow (Swofford *et al.*, 1996).

For large phylogenies, an algorithm propose by Yang (2000) can be used to optimize substitution models. The algorithm consists of two phases. In the first phase BFGS algorithm is used to estimate substitution parameters simultaneously. In the second phase branch lengths are optimized one after another, while substitution parameters are fixed. The procedure continues until a global convergence of all parameters is achieved.

The biggest challenge in phylogeny reconstruction is the discrete optimization because it deals with the search for the optimal tree in a search space that grows exponentially with the number of sequences. In the next chapter we will discuss the metaheuristic Iterated Local Search which is used as the search framework for our tree search algorithm presented in chapter 4.

Chapter 3

Iterated Local Search

In this chapter, we introduce the theoretical background and methodology of the metaheuristic Iterated Local Search (ILS) (Lourenco *et al.*, 2003). The underlying principles and different components of ILS will be discussed in detail. These are the foundations for our tree search framework described in chapter 4.

3.1 Overview

Nowadays, extremely hard computational problems exist virtually in all fields of modern science, ranging from molecular biology to physics and operations research. Most of the problems are combinatorial optimization problems, in which the number of possible solutions grows exponentially with the problem size. Solving large instances of such problems using straightforward exhaustive search strategies is computationally impractical. Heuristic search techniques are therefore developed to speed up the process of finding feasible solutions, while maintaining reasonable accuracy. Optimality, however, is never guaranteed. The majority of heuristic techniques are often based on problem-specific knowledge. As a result, they tend to be ad-hoc and specialized. Thus, such heuristics are hardly transferred to other problems.

Metaheuristics, on the other hand, are developed as general procedures, in which the embedded heuristic for a specific problem can be treated as a "black box" routine. As a result, metaheuristics are applicable to a wide range of problems. ILS is a metaheuristic that belongs to the class of global optimization algorithm. The aim of

global optimization is to search for globally optimal solutions throughout the whole search space. The opposite of global optimization is local optimization, where the locally optimal solution is searched within a constrained region of the search space. The essence of ILS is the construction of a sequence of locally optimal solutions by perturbing the current local optimum and subsequently applying *local search* to the perturbed solution in an iterative fashion (Battiti *et al.*, 2008). When the local search is given as a "black box", ILS acts as the coordinator that guides the local search through the search space to search for better solutions.

3.2 Architecture of ILS

Here we use the same notations as in (Lourenco *et al.*, 2003). Assuming that a problem-specific heuristic is given. We call this routine *LocalSearch* and suppose that it is deterministic. Let $\mathcal{C}$ be the cost function of the combinatorial optimization problem which we want to minimize. We denote by $\mathcal{S}$ the solution space, i.e. the set of all candidate solutions s. *LocalSearch* then defines a many to one mapping from $\mathcal{S}$ to the subset $\mathcal{S}^* \subset S$ of local optimal solutions s^*. For illustration, one could imagine each solution s^* having an "attraction basin", which contains a collection of solutions s that are mapped to s^* by *LocalSearch* (Figure 3.1).

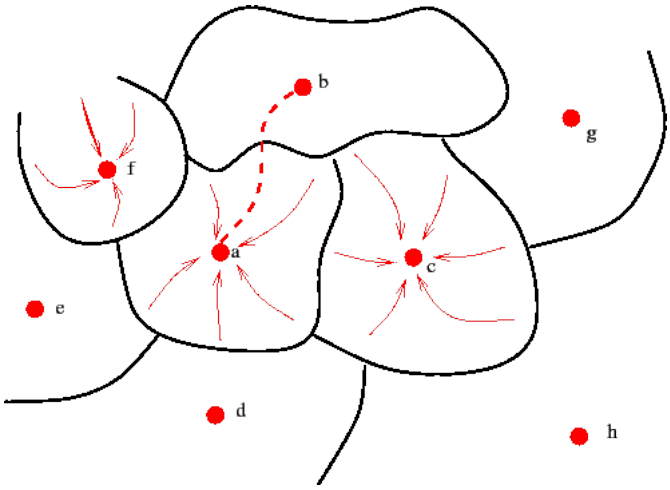

Figure 3.1: Attraction basins of different local optima (Battiti *et al.*, 2008)

The global optima are located within the set $\mathcal{S}^*$. A naive search strategy for

finding the global optima could be *random restart*, in which *LocalSearch* is applied to random initial solutions in $\mathcal{S}$ repeatedly. However, for large problem instances, the size of $\mathcal{S}$ could be too huge for random restart to be useful. Johnson and McGeoch (1997) and Schreiber and Martin (1998) indicated that random restart on large instances produces solutions, whose costs are arbitrarily distributed above the minimal cost. Therefore, random restart has very low probability of finding solutions that are near the global optima. To increase the probability of finding the global optima, we need a biased sampling of $\mathcal{S}^*$ in such a way so as to prefer local optima with the lowest cost.

In order to have a biased sampling, one need to devise a mechanism of how the elements in $\mathcal{S}^*$ are accessed. This requires the definition of the neighborhood structure in $\mathcal{S}^*$. Constructing a neighborhood in $\mathcal{S}^*$ usually demands multiple executions of *LocalSearch*, which is computationally expensive (Lourenco *et al.*, 2003). It is thus of desire if one could explore $\mathcal{S}^*$ without having to explicitly define the neighborhood in $\mathcal{S}^*$. In ILS biased sampling of the set $\mathcal{S}^*$ is achieved heuristically by the construction of a walk from a solution s^* to a "nearby" one, which works as follows. Given a current solution s^*, s^* is perturbed into $s' \in \mathcal{S}$. *LocalSearch* is then applied to s' to obtain a locally optimal solution $s^{*'} \in \mathcal{S}^*$. Then, an acceptance criterion decides whether $s^{*'}$ should be the next element of the walk or not. If not we step back to s^* and repeat the procedure (Lourenco *et al.*, 2003). The procedure is illustrated in Figure 3.2. In this way moving to a neighboring solution only requires one execution of the perturbation method and *LocalSearch*.

ILS consists of four core components: *ConstructInitialSolution*, *LocalSearch*, *Perturbation*, *AcceptanceCriterion*. The performance of ILS can be improved by independently tuning each component. This modular nature of ILS makes the tuning of the algorithm easier than in other less modular metaheuristics (e.g. Genetic Algorithm). The high-level architecture and the general procedure of ILS can be summarized in Algorithm 1. In the following, we will discuss the different aspects of these components.

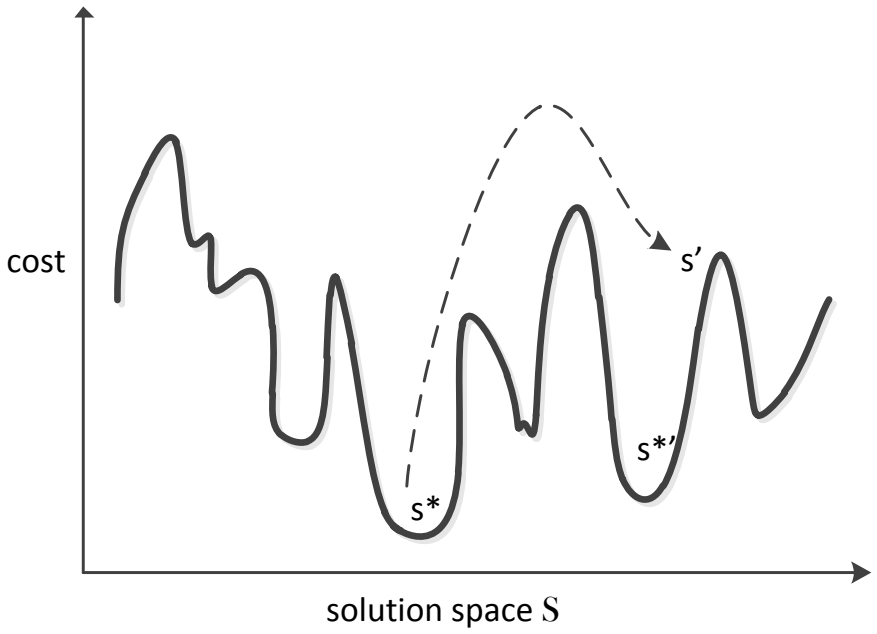

Figure 3.2: Illustration of Iterated Local Search (Lourenco *et al.*, 2003).

Algorithm 1: Pseudocode for Iterated Local Search (Lourenco *et al.*, 2003).

Input: ProblemInstance

Output: s^*

$s_0 \leftarrow$ `ConstructInitialSolution()`;

$s^* \leftarrow$ `LocalSearch(`s_0`)`;

while $\neg$ `StopCondition()` **do**

  $s' \leftarrow$ `Perturbation(`s^*, SearchHistory`)`;

  $s^{*'} \leftarrow$ `LocalSearch(`s'`)`;

  $s^* \leftarrow$ `AcceptanceCriterion(`s^*, $s^{*'}$, SearchHistory`)`

end

return s^*;

3.3 Tuning ILS

3.3.1 Initial solution

Initial solutions can be generated randomly or by a greedy algorithm. Having a good initial solution is important if one wants to reach high-quality solutions as fast as possible. In comparison with random starting solutions, using greedy algorithm often leads to better results much faster when combined with the local search. Hence, the construction method for the initial solution is usually based on greedy strategies.

3.3.2 Perturbation

Perturbation is the main ingredient of the biased sampling mechanism of ILS. In order to have an effective biased sampling where local optima of good quality have high chance of being selected, the perturbation needs to be neither too strong or too weak. If the perturbation is too weak, the local search will likely bring s' back to s^*. If the perturbation is too strong ILS will behave like a random restart algorithm (Lourenco *et al.*, 2003).

The strength of a perturbation is defined as the number of modified solution components. For example, in the Traveling Salesman Problem the number of edges that are modified by the perturbation represents the perturbation strength. Depending on the problem, an appropriate perturbation strength can either be fixed or based on the instant size. Moreover, a more advanced approach to choosing perturbation strength called *adaptive perturbation* can also be applied (Lourenco *et al.*, 2003). In adaptive perturbation the perturbation strength is adapted during the execution of ILS. For example, when the search seems to get stuck, we increase the perturbation strength and continue until a new better solution is found. The perturbation strength is then set back to the initial value.

Generally, the perturbation should be defined in a way so that the local search is unlikely to undo it. To achieve this, apart from avoiding perturbation whose strength is too weak, it is also important not to use the same neighborhood structure as that of the local search. The perturbation can also make use of the characteristics of the problem to produce perturbed solutions of good quality, based on which better

results can be obtained by the local search.

3.3.3 Acceptance criterion

In ILS a random walk within $\mathcal{S}^*$ is performed. The walk is guided by the local search and the perturbation method, which creates a move between the current solution s^* to a neighboring solution $s^{*'}$. Upon reaching a new solution $s^{*'}$, the acceptance criterion is used to decide whether the walk should continue with $s^{*'}$ or not. If $s^{*'}$ is not accepted, we return to the previously visited solution s^*. The acceptance criterion is based on the quality of $s^{*'}$, with respect to the quality of s^*. If the search only accepts better solutions then its behavior is referred to as *intensification* (Equation 3.1).

$$AcceptanceCriterion(s^*, s^{*'}) = \begin{cases} s^{*'} \text{ if } \mathcal{C}(s^{*'}) < \mathcal{C}(s^*) \\ s^* \text{ otherwise.} \end{cases} \tag{3.1}$$

Intensification means that the search focuses intently on the region where the current best solution are previously found. The opposite of intensification is *diversification*, in which new solutions are always accepted regardless of their qualities (Equation 3.2). This means better solutions are searched in different regions of the search space.

$$AcceptanceCriterion(s^*, s^{*'}) = s^{*'} \tag{3.2}$$

In some problems it was shown beneficial to have strategies that balance out the effects of intensification and diversification. For example, Martin *et al.* (1992) implemented a simulated annealing (Kirkpatrick, 1984) type acceptance criterion, where $s^{*'}$ is always accepted if it is better than s^*. If $s^{*'}$ is worse than s^*, $s^{*'}$ is accepted with probability $e^{(\mathcal{C}(s^*)-\mathcal{C}(s^{*'}))/T}$, where T is the temperature parameter which decreases toward the end of the run as in simulated annealing.

3.3.4 Local Search

So far the local search has only been mentioned as a black box routine. Although ILS could work well without the knowledge of the local search, having access to the

internal mechanism of the local search can bring a lot of improvement. The choice of the local search algorithm has a great impact on the overall performance of ILS. For a given combinatorial optimization problem, a number of different local search algorithms are usually available. With respect to the quality of the solutions, it is usually true that the better the local search, the better the corresponding ILS (Johnson and McGeoch, 1997; Stuetzle and Hoos, 1999). However, such local search often requires more computing time. Hence, while choosing an embedded heuristic, one should take into account the trade-off between the quality of solutions against the computing time.

Although the *search history* is not mandatory for ILS, its usage can greatly improve the efficiency of the search. Stuetzle (1999) showed that incorporating search history enhances performance. In chapter 5 we will describe our heuristic search which employs the search history to increase search efficiency.

Chapter 4

Searching For The Optimal Phylogeny

Having described the fundamental concepts of phylogenetic inference and the core techniques in ILS, we are now ready to employ ILS in the context of phylogenetic inference. In this chapter we present our general tree search framework based on ILS. Each component of the framework will be subsequently described in detail.

4.1 General Framework

Our ILS-based tree search framework is depicted in Figure 4.1. This framework was originally proposed by (Vinh and von Haeseler, 2004), albeit not explicitly mentioned as an ILS approach. First, an initial tree is generated. Then the parameters of the substitution model and the branch lengths of the initial tree are optimized. Nearest Neighbor Interchange (NNI) is the tree rearrangement technique used in the local search to move between neighboring solutions. We refer to the local search as *NNI search*. The Important Quartet Puzzling (IQP) (Vinh and von Haeseler, 2004) method is used to perturb the tree obtained from the NNI search. Subsequently, the NNI search is applied to the perturbed tree to generate a new tree. For the acceptance criterion the intensification strategy is used. If the likelihood of the new tree is better than that of the current best tree, the new tree is selected as the new current best tree. As in the ILS framework, this procedure iterates until the stop

condition is met. We refer to each of this iteration as *IQ-Tree iteration* (Figure 4.1). The stop condition is either a predefined number of iterations or a statistical stopping rule that predicts the remaining number of iterations needed for the search, based on how often a new better tree is found (Vinh and von Haeseler, 2004).

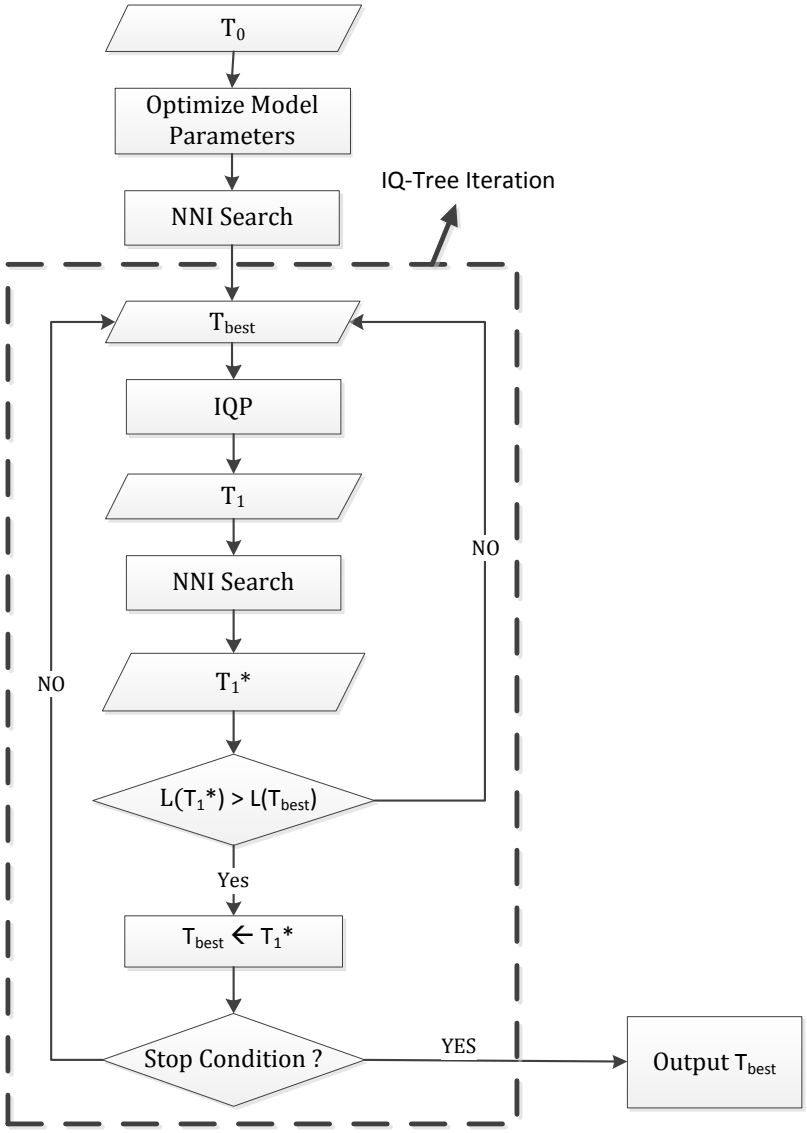

Figure 4.1: The Tree Search Framework

4.2 Initial Tree

As discussed in section 3.3.1, the starting tree can be either randomly generated or built by a greedy algorithm whereas greedy algorithm is usually preferred. Here, for the generation of an initial tree, we use the greedy algorithm BIONJ (Gascuel, 1997), which is an improved version of the neighbor-joining (NJ) algorithm (Saitou and Nei, 1987). By using this approach a good starting tree can be rapidly constructed.

After the initial tree has been generated, the likelihood of the tree is computed and subsequently the model parameters are optimized. For the optimization of the parameters of the substitution model, we use BFGS's method proposed in (Minh, 2005). The branch lengths of the tree are optimized by Newton's method (see 2.4.3). Each branch length is optimized separately, while the others are fixed. This procedure is carried out repeatedly until no improvement of the tree likelihood is observed. During the tree search procedure, the substitution model parameters are fixed and only branch lengths are updated.

4.3 Local Search

In phylogeny search, in order to move from one tree topology to another in the neighborhood, three common tree rearrangement techniques are usually used (Felsenstein, 2004):

- Nearest Neighbor Interchange (**NNI**): Two NNI operations are possible for every branch of the tree (see Section 4.3.1).
- Subtree pruning and regrafting (**SPR**): A subtree is pruned from the tree and then redrafted to a different location on the tree.
- Tree Bisection and Reconnection (**TBR**): The tree is bisected along a branch, generating two disjoint subtrees. The subtrees are then reconnected by creating an addition branch connecting the two subtrees.

The neighborhood sizes correspond to the three aforementioned tree rearrangement techniques are: $O(n)$ (NNI), $O(n^2)$ (SPR), $O(n^3)$ (TBR), where n is the

number of leaves (sequences) in the tree (Felsenstein, 2004). The larger the size of the neighborhood is, the longer it takes to evaluate all neighboring solutions. Since the effectiveness of ILS attributes to the rapid sampling of the set of local optima, it is essential for the local search to be fast. Therefore, we chose NNI as our tree rearrangement technique because it has the smallest neighborhood size. In section 4.3.3, we will present different steps involved in our NNI search.

4.3.1 Nearest Neighbor Interchange

In Figure 4.2, the tree T_1 on the left-hand side has four subtrees W, X, Y and Z. With respect to the internal branch (a, b), the two possible NNI operations are: swapping X with Y and swapping X with Z (due to symmetry, swapping W with Z or W with Y will result in the same tree as swapping X with Y or X with Z, respectively). T_2 and T_3 are the resulting *NNI-trees* of the two NNI operations. For a tree with n leaves, there are $n - 3$ internal branches, thus the total number of NNI-trees is $2n - 6$.

4.3.2 Branch Length Optimization

Branch length optimization is an essential part of maximum likelihood phylogeny reconstruction because a phylogeny is not only defined by the tree topology but also its branch lengths. Here we present the computation needed to determine the optimal length of a branch. This involves the likelihood computation of the tree.

Let us now consider the optimization of the branch length $t_{a,b}$, where a and b are the two nodes of the branch. We denote by $L_a(x_a)$ the partial likelihood of the subtree with root a, whose character at the site is x_a. The a priori probability of character x_a is denoted as π_{x_a}. $\mathbf{P}_{x_a x_b}(t_{a,b})$ is the probability for character x_a to become x_b in time $t_{a,b}$. The tree likelihood at site i is computed as follows (Yang, 2000):

$$\mathbf{L}_i = \sum_{x_a \in \{A,C,G,T\}} \sum_{x_b \in \{A,C,G,T\}} \pi_{x_a} \mathbf{P}_{x_a x_b}(t_{a,b}) \mathbf{L}_a(x_a) \mathbf{L}_b(x_b). \tag{4.1}$$

Here the computation is described for DNA only, but it can also be used with proteins by changing the character set. The likelihood of the whole tree is equal to

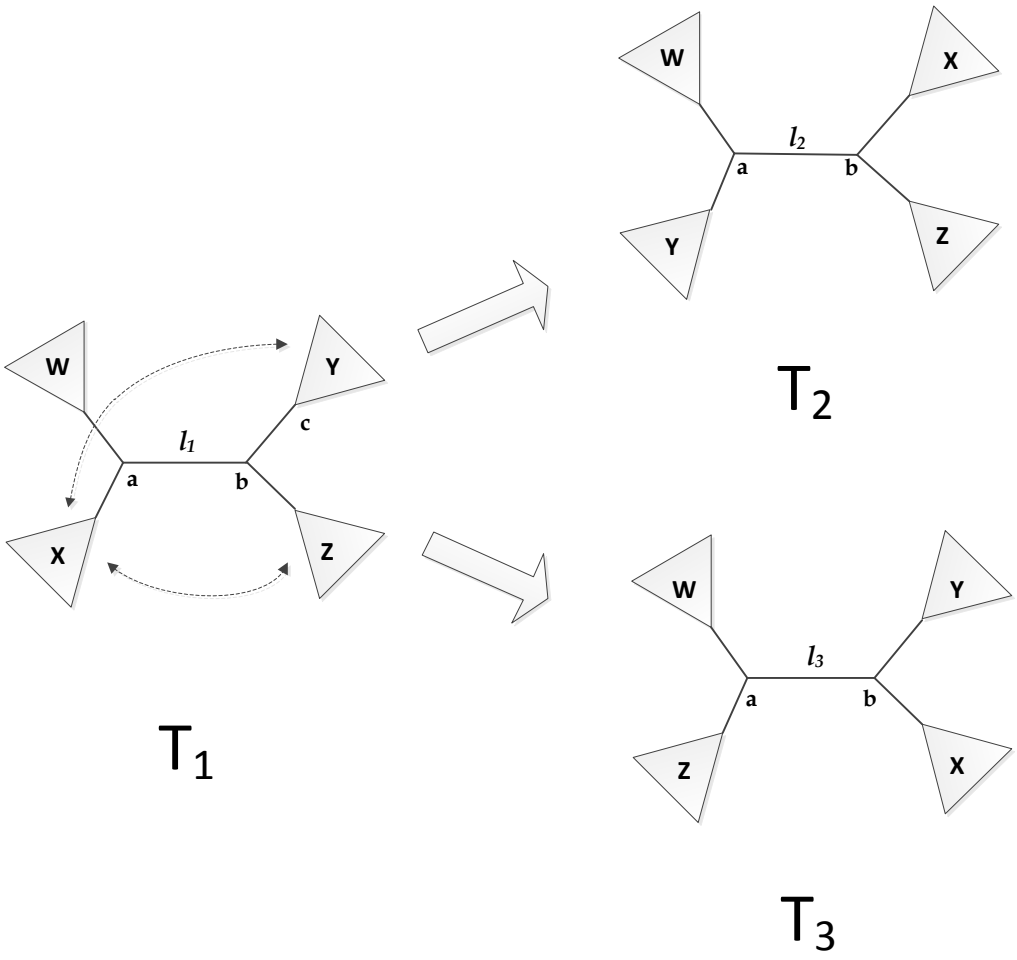

Figure 4.2: NNI operations.

the product of the site likelihoods:

$$\mathbf{L}_{tree} = \prod_i \mathbf{L}_i. \tag{4.2}$$

The optimal value of $t_{a,b}$ is the one that maximizes $\mathbf{L}_{tree}$, according to Equation 4.2. To optimize $t_{a,b}$, we use Newton's method for one parameter function.

4.3.3 NNI Search

A common strategy to search for the local optimal tree by NNI is to first evaluate all $2n - 6$ NNI-trees. Then, for each NNI-tree all the branch lengths are optimized

to obtain the optimal tree likelihood. NNI-tree with the highest likelihood will be chosen as the tree for the next move in the local search. This procedure is repeated until the local optimum is found.

The aforementioned strategy requires the optimization of all the branch lengths of every NNI-tree to determine the best one. This is a computationally intensive process and thus is very slow. Guindon and Gascuel (2003) introduced an efficient heuristic using NNI that is able to find the local optimal tree in short time. The essentials of the heuristic lie in the way it handles branch length optimization and the application of NNIs. The heuristic is used in our NNI search algorithm. In the following, we will present details of the heuristic.

Evaluating NNI-trees

To compare the likelihoods of the two NNI-trees with that of the current tree, instead of optimizing all the branch lengths of each tree, only the length of the internal branch corresponding to the NNI is optimized. Let L_{T_1} be the likelihood of the current tree T_1 after the optimal length l_1 of branch (a, b) has been determined. Similarly, L_{T_2} and L_{T_3} are the likelihoods of the two NNI-trees T_2 and T_3 after computing the optimal lengths l_2, l_3 (Figure 4.2). If L_{T_2} is larger than L_{T_1} and L_{T_3}, then tree T_2 is better than the two other trees. In this case, we call the corresponding NNI operation from T_1 to T_2 a *positive NNI* and define a score $\delta = L_{T_2} - L_{T_1}$ for the positive NNI.

Note that the pruning algorithm, which we use to compute the tree likelihood, is a dynamic programing method. Here, the partial likelihood of a subtree is stored at the root node of that subtree. If the topology of the subtree does not change, its partial likelihood needs not to be recomputed. The precomputed partial likelihoods of the subtrees W, X, Y and Z can therefore be reused for the likelihood computation of the NNI-trees. Thus, computing L_{T_2} and L_{T_3} requires essentially the same time as computing L_{T_1}. Such possibility for fast evaluation of solutions in the neighborhood is very important, otherwise doing a local search would be very computationally expensive.

Performing NNI Operations

Likelihoods of the NNI-trees are first independently computed by optimizing the corresponding branch lengths as described above. Then, the positive NNIs are determined. Subsequently, an appropriate portion of the positive NNIs are simultaneously applied to the tree. Here, by performing many NNIs at the same time instead of one by one, the NNI search can quickly find the local optimum.

The Whole Procedure

Our NNI search procedure consists of the following steps:

1. Evaluate the likelihood of $2n - 6$ NNI-trees of the current tree to identify positive NNIs.

2. When two NNIs correspond to two adjacent branches, they can have one subtree in common. In Figure 4.2, the branches (a, b) and (b, c) of tree T_1 are adjacent and the corresponding NNIs have Z as the subtree in common. If two positive NNIs correspond to two adjacent branches, only NNI with the highest δ is conserved. Thus, we obtain a list K of positive NNIs that do not have joint branches affected by the NNI operations. Elements in K are ranked according to their score δ.

3. We then simultaneously apply a proportion λ of the NNIs in K to the current tree, starting from the higher value of δ.

4. Having $\lambda = 1$ means all the NNIs are simultaneously applied, whereas $\lambda = 0$ means the current tree is left unchanged. Performing a single positive NNI increases the tree likelihood. However, simultaneously applying several positive NNIs to the tree will not guarantee to improve the tree likelihood, since edges are not independent (Guindon and Gascuel, 2003). Fortunately, in majority of the cases, the tree likelihood increases when "most" of the NNIs are applied. Like in (Guindon and Gascuel, 2003), we choose 0.75 as the initial value of λ. In case the tree likelihood decreases, we "roll back" the tree to its original state and reapply a new proportion $\lambda/2$ of the NNIs in K. This procedure is

repeated until the tree likelihood increases. If λ becomes too small so that the corresponding proportion in K is smaller than 1, then only the best NNI in K is performed and the tree likelihood is guaranteed to be increased.

5. Optimize all the branch lengths of the new tree. Here, our approach for branch length optimization differs from that originally described in (Guindon and Gascuel, 2003). In the paper, branch lengths of the new tree are optimized using a heuristic approach to reduce computational time (the reader are referred to the paper for more details). Since we did not notice any performance advantage of this approach, we decided not to use it.

6. Go back to step 1. This is repeated until no positive NNIs can be found.

Steps 1-6 constitute a so-called "NNI round" (Figure 4.3).

4.4 Perturbation Method

For the perturbation step we use the Important Quartet Puzzling (IQP) algorithm proposed by (Vinh and von Haeseler, 2004), which is an extension of the Quartet Puzzling method (Strimmer and von Haeseler, 1996). The tree is perturbed by randomly removing a proportion of the leaves and then re-inserting them in a random order back into the tree, using IQP. In the following we will briefly describe the basic concepts of the two algorithms. For the extensive descriptions we refer the reader to the two aforementioned papers.

4.4.1 The Quartet Concept

A quartet is defined as a tree with four taxa. For a group of four sequence A, B, C and X, there are three possible quartets (Figure 4.4). The quartet tree concept serves as the building block for the Quartet Puzzling (QP) algorithm.

4.4.2 Quartet Puzzling

Quartet Puzzling is a greedy algorithm for reconstructing phylogenetic tree by maximum likelihood. In comparison to other heuristic search strategies which optimize

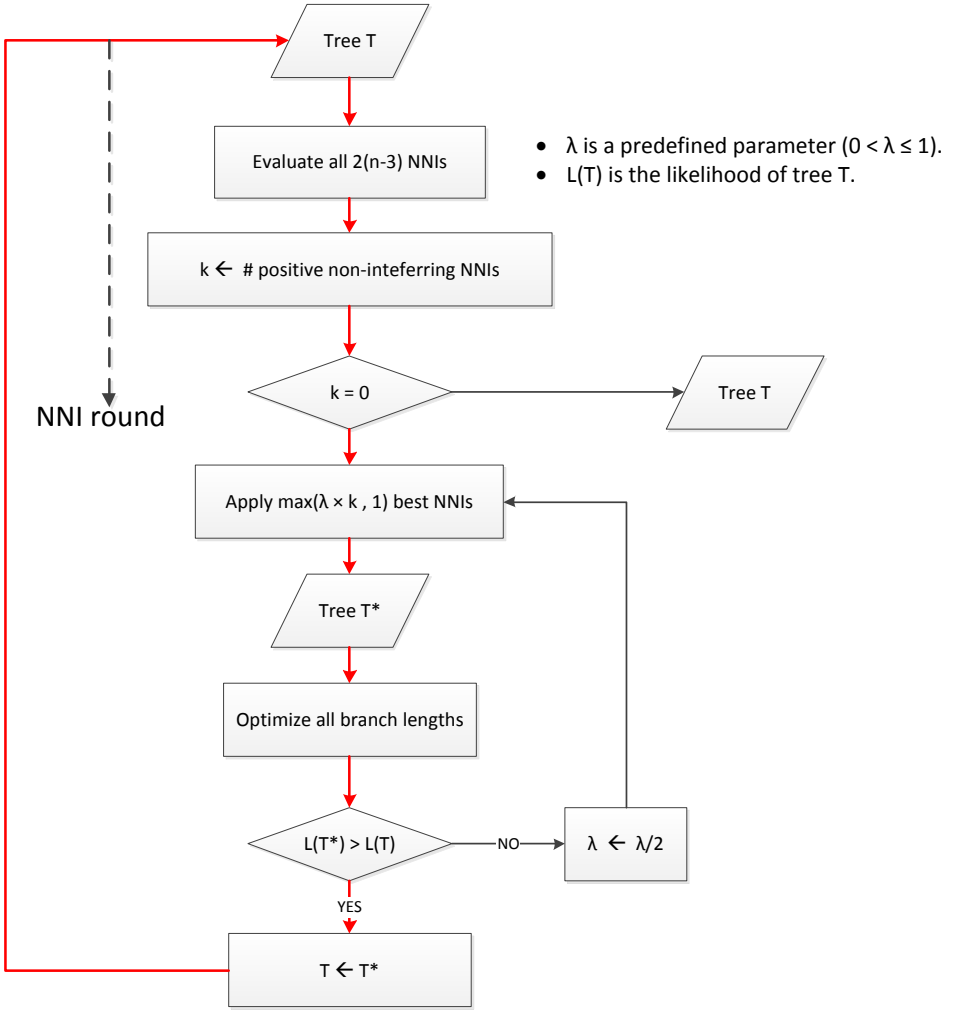

Figure 4.3: Flowchart diagram of the local search.

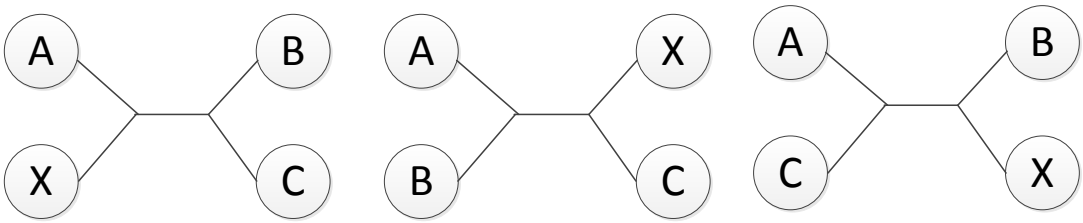

Figure 4.4: Three possible quartets for a group of four sequence A, B, C and X.

an objective function, QP required lower computational efforts. The method first evaluates all $\binom{n}{4}$ quartet trees (n the number of sequences) by their likelihood values to determine the ML quartet tree for any set of four sequences. It then carries out

a so-called puzzling step with the aim of combining the quartet trees to a full tree with n sequences. In the puzzling step, sequences are added sequentially in random order to an existing subtree. With regards to the ML quartet trees, the position of a new sequence is determined by a voting procedure. An intermediate tree of $k < n$ sequences is produced at each puzzling step. The puzzling step is repeated several times and the quartet puzzling tree is obtained by a majority-rule consensus (Margush and McMorris, 1981) of all intermediate trees.

4.4.3 Important Quartet Puzzling

Important Quartet (IQ) is an extended definition of the quartet concept. An IQ also contains four leaves A, B, C and X, where A,B and C are chosen so that they are "closely related" to each other in the current tree and X is the new sequence to be inserted into the tree. The "closely related" concept can be defined by the introduction of the so-called *k-representative leaves*. A set of k-representative leaves of a binary rooted tree T is a collection of at most k [1] leaves that has smallest distances from the root, where the distance is simply defined as the number of branches connecting the two respective nodes (Vinh and von Haeseler, 2004).

Consider an unrooted tree T. The tree can then be rooted at an arbitrary internal node r. The virtual root r then splits T into three rooted disjoint subtrees T_1, T_2, and T_3 (Figure 4.4.3). The sets of the k-representative leaves can be computed for each of these subtrees. The nodes A, B and C of an important quartet can be taken from each of the computed k-representative leaf sets. To put a new sequence X into tree T, the important quartets are determined for all internal nodes of T. Then for each important quartet (A, B, C, X), the optimal tree topology is computed using a fast tree reconstruction method (e.g. Neighbor Joining). From here, the position of the new sequence X can be determined using the same procedure as in the Puzzling step of the Quartet Puzzling method.

[1] In practice, the value chosen for k is much more smaller than the number of sequences, usually 4 or 5.

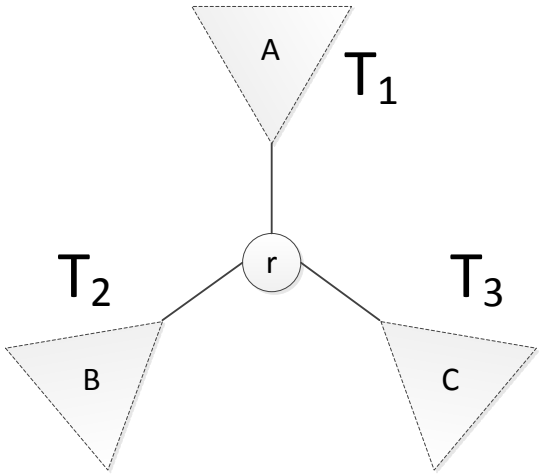

Figure 4.5: Tree can be split by a virtual root into three rooted subtree T_1, T_2 and T_3. Nodes A, B, and C are picked up from the k-representative leaf set of T_1, T_2 and T_3 respectively.

4.4.4 Perturbation With IQP

To perturb the local optimal tree obtained from the NNI search, we first delete each leaf of the tree with probability p_{del} $(0 < p_{del} < 1)$. Then the deleted leaves are reinserted back into the tree using the IQP algorithm. The advantages of using IQP as our perturbation method are twofold. First, by perturbing the tree in a constructive way, the quality of the new tree is higher than those produced by simple randomization. Second, the mechanism IQP uses to create a new perturbed tree is substantially different than that used in the NNI search. This means their neighborhood structures are different from each other and the NNI search will thus unlikely undo the perturbation.

The perturbation strength is regulated by the parameter p_{del}. As in (Vinh and von Haeseler, 2004), we adjust p_{del} according to the size of the input alignment. Let n be the number of sequences contained in the input alignment. p_{del} is then defined as follows:

$$p_{del} = \begin{cases} 0.5 \text{ if } n \leq 50 \\ 0.3 \text{ if } 50 < n \leq 200 \\ 0.1 \text{ if } 200 < n \end{cases} \tag{4.3}$$

Chapter 5

Optimizing The Search

Chapter 4 presents the basis of our ILS based search strategy for finding the optimal phylogenetic tree. In this chapter we will introduce two extensions to our tree search algorithm. In the first extension, we employ search history to help the NNI search avoid non-promising regions, i.e. regions with little chance of encountering better trees (section 5.1). The second extension deals with the vectorization of the computational routines for the calculation of the tree likelihood (section 5.2). Both of these extensions are aimed at speeding up the overall runtime performance of the search.

5.1 Adaptive NNI Search

In our preliminary runs of the IQ-Tree algorithm with several hundred iterations, we observed the following behavior. In the first iterations, the number of times IQ-Tree discovers better trees is very high. Whereas, in the remaining iterations, it takes a significant number of iterations until a better tree is found. This behavior can be explained by the fact that in the late phase, the size of the attraction basins of the local optima (see section 3.2), which are better than the current best one, is getting smaller. Therefore, the probability for the search to reach those attraction basins also becomes lower. This led to our idea of improving the performance of the search by avoiding the basins of attraction of low quality local optima. In this section we will describe our heuristic approach to achieve that idea. From now on, we

want to make clear that when we talk about iteration we refer to IQ-Tree iteration, which contains multiple NNI rounds (see chapter 4). The reader is reminded of the distinction between the two terms: *iteration* and *NNI round*.

To infer the phylogenetic tree from an input alignment, assume that q IQ-Tree iterations are carried out. We denote by n_i the number of NNIs performed in iteration i and by L_i the log-likelihood of the final tree obtained at the end of the iteration, for $i = 1 \dots q$. The log-likelihood of the current best tree after i iterations is then defined as

$$L_i^{best} = \max\{L_1, L_2 \dots, L_i\} \tag{5.1}$$

Each iteration i will start with the current best tree, whose log-likelihood is L_{i-1}^{best}. The tree is then perturbed by IQP, resulting in a tree with log-likelihood $L_{i,0}$. This tree is gradually improved by the application of NNIs.

Figure 5.1 is a plot showing the change in the log-likelihood of the tree with respect to the number of NNIs in different iterations of the same run. The log-likelihood change in each iteration is represented by a colored curve. As seen from the plot, each curve starts with the current best log-likelihood L_i^{best} then drops to the log-likelihood $L_{i,0}$ by the affect of IQP. Subsequently, the curve goes up gradually when the tree is improved by NNIs. The curve ends at the locally optimal log-likelihood L_i. The number of NNIs and the log-likelihoods of the tree are shown in the horizontal and vertical axis, respectively. From the plot we deduced two important features observed within each iteration:

1. The log-likelihood improvement is more or less linear with the number of NNIs.

2. When the log-likelihood $L_{i,0}$ of the perturbed tree is very low, compared to the current best log-likelihood L_i^{best}, it is unlikely that the log-likelihood L_i of the locally optimal tree is better than L_i^{best}. The green curve at the bottom of the plot is such an example.

Based on this observations, we devised a heuristic called *adaptive NNI search* that predicts in every iteration i, after the IQP step, whether the NNI search would lead to a final tree with log-likelihood $L_i > L_i^{best}$. If $L_i \leq L_i^{best}$, according to the

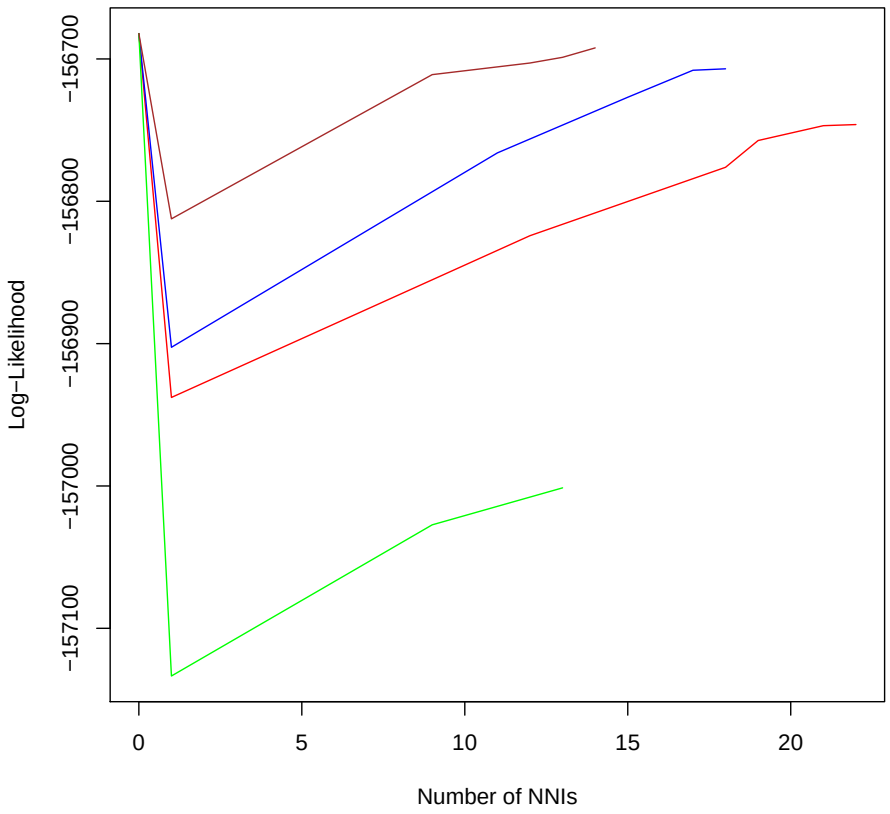

Figure 5.1: Improvements made by NNIs for several iterations.

prediction, the NNI search is skipped and thus saving valuable computation time needed for the execution of the NNI search. Details of the heuristic will be discussed in the following.

Assume that in iteration i, there are k NNI rounds. Let $L_{i,j}$ be the log-likelihood of the intermediate tree obtained at the end of NNI round j, where $1 < j < k$ and $L_{i,0} < L_{i,1} < \cdots < L_{i,k}$. The number of NNIs performed in NNI round j is denoted by $m_{i,j}$ ($\sum_{j=1}^{k} m_{i,j} = n_i$). The average log-likelihood improvement made per NNI δ_{ij} in NNI round j is calculated using

$$\delta_{ij} = \frac{L_{ij} - L_{i,j-1}}{m_{i,j}} \tag{5.2}$$

We now study the distribution of n_i and δ_{ij} for a complete run of the IQ-Tree algorithm with 436 iterations.

Figure 5.2 and Figure 5.3 show the distribution of n_i and δ_{ij}. From the distri-

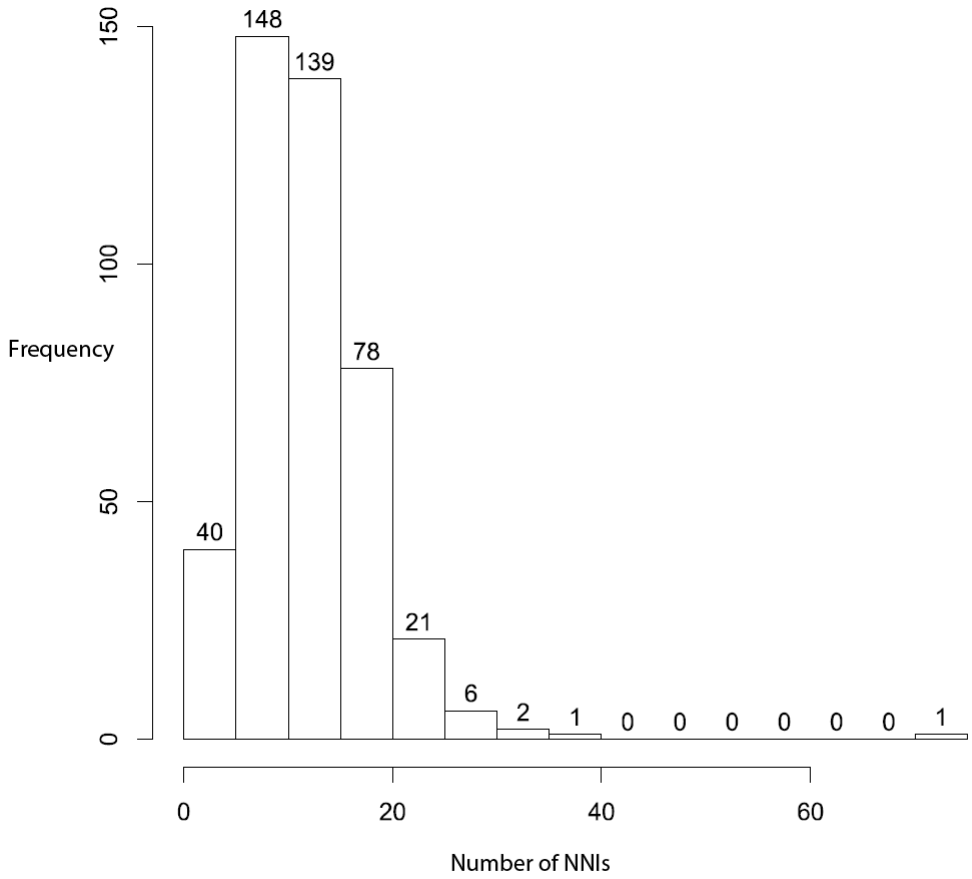

Figure 5.2: Histogram of n_i.

butions we observed that

- n_i and δ_{ij} have maximal frequencies at the values 10 and 11, respectively.
- For $n_i > 10$ and $\delta_{ij} > 11$: the higher the value, the lower its frequency.

We calculate the critical values n_i^P and δ_{ij}^P of the two distributions as a function of P ($1 < P < 99$). Having $P = 95$ means that 95% of the time the values of n_i and δ_{ij} are below n_i^P and δ_{ij}^P, respectively. L_i^P is then defined as the log-likelihood that is greater than L_i P% of the time. For each NNI round k of iteration i, L_i^P is computed as follows.

$$L_i^P = L_{i,k-1} + (n_i^P - \sum_{j=1}^{k-1} m_{i,j}) \times \delta_{i,j}^P \tag{5.3}$$

Our adaptive NNI search now contains the following steps:

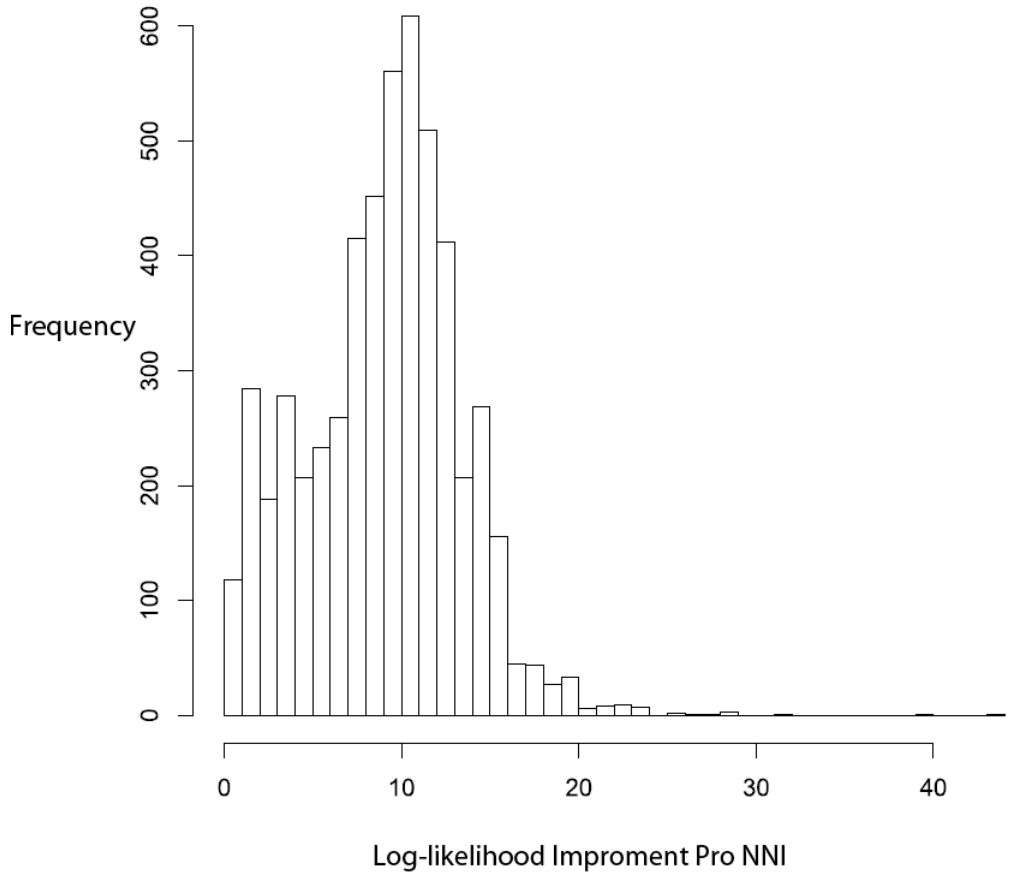

Figure 5.3: Histogram of δ_{ij}.

1. The first 100 iterations are performed as described in section 4.1. In addition, we keep track of n_i and δ_{ij} in each iteration.

2. After the first 100 iterations, we create the empirical distributions of n_i and δ_{ij}. The distributions are kept updated by adding new values of n_i and δ_{ij} in the remaining iterations.

3. At the beginning of each NNI round k of iteration i ($i > 100$), we compute L_i^P according to Equation 5.3 where n_i^P and δ_i^P are computed from the empirical distributions (P is a predefined confidence level).

4. We abort the NNI search in iteration i and continue with iteration $i + 1$ if $L_i^P \leq L_i^{best}$.

From an ILS perspective, here the search history (the statistics of n_i and δ_{ij}) is

incorporated into the process of searching for the optimal tree. This helps the search avoids regions, where better local optima can unlikely be found. The numerical results will be presented in section 5.3.

5.2 Parallelization Of The Likelihood Computation

In the previous section we have described a heuristic technique to speed up the search by avoiding search regions that will not lead to better trees. Parallelization is another approach to reduce computation time and is commonly used when algorithmic improvements have reached their limits. According to our profiling benchmark, the computation of the tree likelihood consumes almost 90% of the running time. Hence, we focus our attempts on parallelizing the likelihood function. In this section, we will present the application of parallelization to the computation of the likelihood. The specific technique we use for parallelization is termed *vectorization*.

5.2.1 Vectorization

In parallel computing vectorization belongs to the class of *data level parallelism*, in which software programs that only perform one operation on a single data (*scalar* implementation) are modified to perform the same operation on multiple data simultaneously (*vector* implementation). This is achieved by utilizing the vector instructions of the Central Processor Unit (CPU) (Hillis and Steele, 1986). The instruction set for vectorization is commonly referred to as *SIMD* (Single instruction, multiple data) instruction set (Flynn, 1972). The SIMD implementation of the conventional CPUs produced by Intel and AMD is called *SSE* (Streaming SIMD Extensions)

The main advantage of SIMD is that when mathematical calculations on large set of a data points are needed, SIMD instructions can perform these operations much faster than non-SIMD instruction. Figure 5.4 shows an example of SIMD operation applied to the addition of two vectors of size four. Only one SIMD addition instruction is needed for the computation, whereas it requires four normal addition

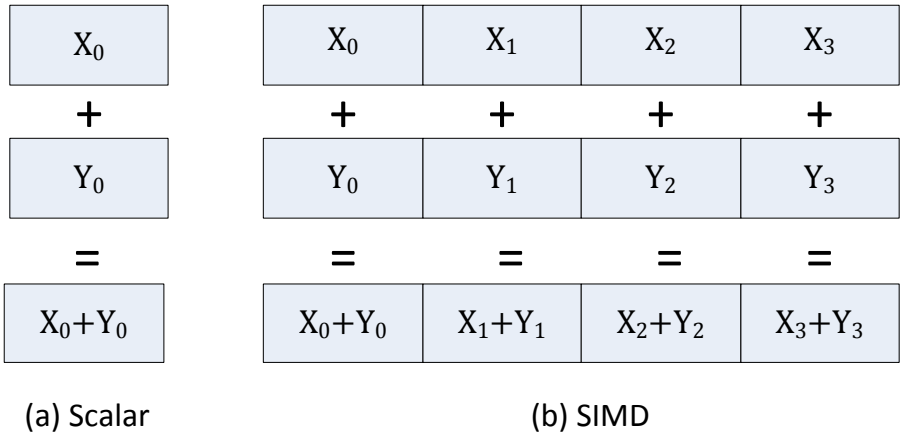

Figure 5.4: Scalar vs. vector implementation.

instruction in a scalar format. The number of data elements that can be processed in parallel depends on the size of the elements and the processing capacity of the processor. In conventional desktop computers SIMD instructions can simultaneously do computations on four single precision numbers or two double precision numbers.

However, there are challenges with the application of SIMD technology. Although auto-vectorization is an active area in compiler research, the most advanced compilers are only capable of vectorizing simple algorithms. Thus, implementing an algorithm with SIMD usually requires a lot of efforts involving low-level programming. Moreover, SIMD instructions are architecture and application specific. Re-implementation is therefore usually required if one wants to extend the application.

SIMD technology are widely used in the fields of multimedia processing, computer graphic and scientific computing. With the rise of general-purpose computing on graphics processing units (GPGPU), application of SIMD is becoming more and more popular. The typical application for SIMD instructions is linear algebra. Nowadays, numerous SIMD-optimized linear algebra software libraries are available, such as Mathlab (**Mat**rix **Lab**oratory from Mathworks), Intel's MKL (Intel's **M**ath **K**ernel **L**ibrary) and ACML (**A**MD **C**ore **M**ath **L**ibrary).

5.2.2 Vectorizing The Likelihood Computation

Having considered all the challenges concerning the application of the SIMD technology, we concluded that using an existing linear algebra library would be the best choice for vectorizing our likelihood function. First, it relieves us from the complication of implementing the algorithm in the low-level SIMD instructions, which can leads to worse performance if it is not done appropriately. Seconds, using a linear algebra library will make the code portable since changes and differences in computer architectures are taken care of by the developers of library.

However, in order to apply such library in our program, we need to do the likelihood computation by means of matrix calculations. By doing this, the code can also be exposed to a more advanced parallelization technology like GPGPU. In the following we present the transformation of the likelihood computation formula into that of the corresponding matrix operations.

For convenient reading, we rewrite the formula (Equation 4.1) for computing the tree likelihood at site i, given an internal branch (a, b) .

$$L_i = \sum_{x_a \in \{A,C,G,T\}} \sum_{x_b \in \{A,C,G,T\}} \pi_{x_a} P_{x_a x_b}(t_{a,b}) L_a(x_a) L_b(x_b). \tag{5.4}$$

For each node a containing character x_a at the current site, the partial likelihood $L_a(x_a)$ can be stored as element of vector $\mathbf{L}_a$. The size of $\mathbf{L}_a$ is equal to the number of possible characters represented by x_a. For example, if we use DNA data then the size of $\mathbf{L}_a$ is 4. For protein data $\mathbf{L}_a$ will have size of 20.

The value of the product $\pi_{x_a} P_{x_a x_b}(t_{ab})$ is the element of the transition matrix $\mathbf{P}$ (see 2.4.1). For DNA, A is a 4×4 matrix, whereas in case of protein A is a 20×20 matrix.

Equation 5.4 can therefore be rewritten as follows:

$$L_i = (\mathbf{L}_b \mathbf{P}) \cdot \mathbf{L}_a \tag{5.5}$$

$\mathbf{L}_b\mathbf{P}$ is the vector-matrix product of vector $\mathbf{L}_b$ and matrix $\mathbf{P}$, whereas $(\mathbf{L}_b\mathbf{P}) \cdot \mathbf{L}_a$ is the dot product of the resulting vector from $\mathbf{L}_b\mathbf{P}$ and vector $\mathbf{L}_a$.

Equations 5.5 can now be vectorized through the usage of the corresponding matrix computation routines in a SIMD linear algebra library.

We chose the linear algebra software library EIGEN (http://eigen.tuxfamily.org) to vectorize our likelihood computation routines. In contrast to the other three libraries mentioned above, EIGEN is an open source project whose performance is at the same level with those of the commercial software.

5.3 Experimental Results

For the purpose of our work we developed a program called IQ-Tree. The program was implemented based on the source code of the Phylogenetic Diversity Analyzer (PDA) software written by Minh *et al.* (2006). The source code was written in the C++ programming language and is available upon request.

In our analysis, we first compared the accuracy of trees reconstructed by IQ-Tree with those by IQPNNI on simulation data (section 5.3.1). Then, we used real data to compare the runtime performance and the log-likelihood of trees produced by the two programs (section 5.3.2). All analyses were carried out on the same computer with CPUs of type *AMD Opteron Processor 6136*.

5.3.1 Simulated Data

Since all phylogeny reconstruction methods are only developed based on assumptions about the evolutionary process, the reconstructed trees are not guaranteed to exactly reflect the evolutionary relationships among the species. However, one can still measure the accuracy of the methods by using simulated data. This works as follows. First, a random phylogenetic tree is generated. Then, the evolution of nucleotide or amino acid sequences is simulated along the tree. The result of this simulation is a sequence alignment of the contemporary species. Based on this sequence alignment a phylogenetic tree is then reconstructed. The accuracy of the phylogeny reconstruction method is verified by comparing the reconstructed tree with the original one. There are different metrics to compare a pair of phylogenetic trees. One of the most widely used metrics is the Robinson-Foulds (RF) distance (Robinson *et al.*, 1981).

The RF distance between two trees is computed as the number of partitions

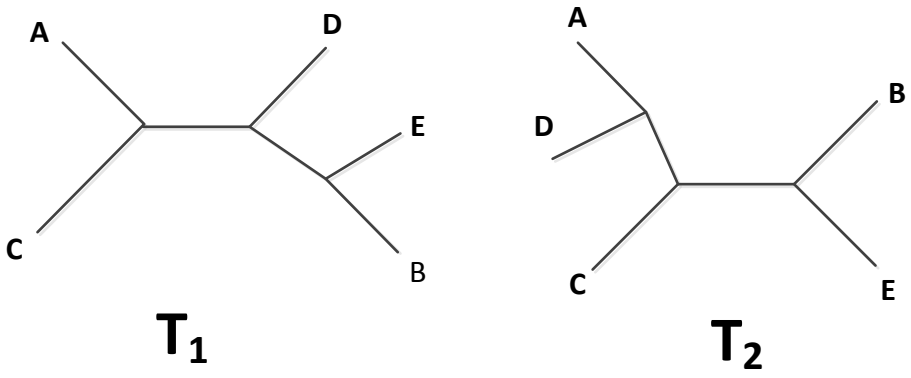

Figure 5.5: Comparison of two phylogenetic trees.

that are contained in one tree and not in the other. The smaller the RF distance is, the more similar the two trees are. Figure 5.5 shows an example of two simple phylogenetic trees. Tree T_1 has two internal branches, which induce two partitions $\{A, C|B, D, E\}$ and $\{A, C, D|B, E\}$. Similarly, tree T_2 also has two partitions $\{A, D|B, C, E\}$ and $\{A, C, D|B, E\}$. T_1 and T_2 share the same partition $\{A, C, D|B, E\}$, whereas $\{A, C|B, D, E\}$ is only contained in T_1 and $\{A, D|B, C, E\}$ is only contained in T_2. Thus, the RF distance between T_1 and T_2 is 2 (Felsenstein, 2008).

In the following we will compare the accuracy of phylogenetic trees generated by IQ-Tree and IQPNNI for DNA and protein data by means of simulated data.

For DNA data, we generated 20 random trees using PDA. Each tree contains 200 leaves. Then we used the program Seq-Gen (Rambaut and Grass, 1997) to emulate the evolutionary process along each tree, according to the HKY model (Hasegawa *et al.*, 1985). The sequence length was 1000. The resulting 20 sequence alignments were used by IQ-Tree and IQPNNI to reconstruct the phylogenetic trees. Subsequently, the trees obtained from IQ-Tree and IQPNNI were compared to the original trees by calculating the RF distance between them. For the computation of RF distance we used the program TREEDIST in the PHYLIP package (Felsenstein, 1993). Finally, we computed the average RF distance of trees reconstructed by each program and compare those values with each other.

We did similar simulation for protein data, in which 20 random trees were also

Table 5.1: Comparison of average RF distances by IQ-Tree and IQPNNI.

Data type	Average RF distance	
	IQ-Tree	IQPNNI
DNA	13.7	13.6
Protein	5.0	5.1

generated. Each tree contains 100 leaves with sequence length of 1000 and WAG (Whelan and Goldman, 2001) was chosen as the model of sequence evolution.

IQ-Tree and IQPNNI were both run with 400 iterations. The P parameter in IQ-Tree (section 5.1) is set equal 0.95 (the default value). The results are shown in Table 5.1. As we can see from the table, the average RF distance of trees produced by IQ-Tree and IQPNNI are quite similar. The differences in the average RF distances by the two programs are not substantial because of the stochastic nature of the algorithms. Thus, for simulated data both methods perform equally well.

5.3.2 Real Data

We used four real data (two DNA and two protein sequence alignments - Table 5.2) to compare the runtime performance and the log-likelihood of the reconstructed trees. The two DNA alignments were both used by Vinh and von Haeseler (2004) and Guindon and Gascuel (2003) to demonstrate their algorithms in the respective papers. The published results are not relevant for our comparison since the current version of IQPNNI outperforms them by a large margin, both in term of runtime and tree log-likelihood. The two protein alignments 74AA and 144AA were kindly provided by Dr.Heiko A.Schmidt and Dr.Ingo Ebersberger, respectively. We used the HKY model (Hasegawa *et al.*, 1985) for the DNA sequences and the WAG model (Whelan and Goldman, 2001) for the protein sequences.

Table 5.2: Data used for runtime analyses.

Type	Name	Number of Sequence	Sequence Length	Model
DNA	218DNA	218	4182	HKY
DNA	500DNA	500	1398	HKY
Protein	74AA	74	4013	WAG
Protein	144AA	144	449	WAG

In order to measure the speedup made by each runtime improvement technique (section 5.1 and section 5.2), we differentiate among three variants of IQ-Tree:

- IQ-Tree^{sse}: Only vectorization is enabled and adaptive NNI search is disabled.
- IQ-Tree$^{adapt-nni}$: Only adaptive NNI search is enabled and vectorization is disabled.
- IQ-Tree: The default program with adaptive NNI search and vectorization enabled.

On all data IQ-Tree and its variants were run with 400 iterations. The same applies for IQPNNI. For each data we ran the programs 10 times and calculated the average runtime. In addition to the average runtime, we also show the best log-likelihood scores obtained by IQPNNI and IQ-Tree after the 10 runs on each data.

Table 5.3: Speedup for vectorization.

Data	IQPNNI Runtime	IQ-Tree^{sse} Runtime	Speedup
218DNA	5296.3s	2863.6s	1,85x
500DNA	7323.7s	4161.5s	1,76x
74AA	20158.5s	7585.1s	2,65x
144AA	15309.8s	4237.8s	3.61x

Table 5.3 shows the speedup made by the application of vectorization. Here we can see the speedup factor for DNA data is about 1.8x. Whereas for protein data

the speedup factor is substantially higher. This can be explained by the intensive computation needed when computing the tree likelihood on protein sequences. Such computation can largely benefit from the vectorization technique.

Table 5.4: Speedup for adaptive NNI search.

Data	IQPNNI Runtime	IQ-Tree$^{adapt-nni}$ Runtime	Speedup
218DNA	5296.3s	3405.5s	1.56x
500DNA	7323.7s	4704.5s	1.56x
74AA	20158.5s	12295.2s	1.64x
144AA	15309.8s	7617.8s	2.0x

The adaptive NNI search also showed significant runtime improvement (Table 5.4), although the speedup factors are a little bit lower than those from the vectorization. It should be noted that the adaptive NNI search is only activated after iteration number 100. This means the larger the number of iteration is, the more effective the adaptive NNI search will be.

Table 5.5: Speedup and best log-likelihood comparison between IQPNNI and IQ-Tree

Data	IQPNNI Runtime / Log-likelihood	IQ-Tree Runtime / Log-likelihood	Speedup
218DNA	5296.3s / -156574.96	2092.3s / -156557.71	2.53x
500DNA	7323.7s / -100004.94	3183.2s / -99997.94	2.3x
74AA	20158.5s / -184996.1	6041.9s / -184996.1	3.34x
144AA	15309.8s / -24644.20	3570.9s / -24643.55	4.29x

When the two runtime improvement techniques are enabled, depending on the data IQ-Tree ran from two to four times faster IQPNNI (Table 5.5). For the protein data the speedup factor is remarkably high (3.34x for 74AA and 4.29x for 144AA). Interestingly, the best tree log-likelihoods found by IQ-Tree are also higher than

those found by IQPNNI in three out of four data (218DNA, 500DNA and 144AA). For the 74AA data, both IQ-Tree and IQPNNI show the same best log-likelihood. The RF distance computed between the two trees also showed that they are identical.

In order to have an insight into how the tree is improved within the 400 iterations by both programs, we created plots that illustrate the log-likelihood improvement with respect to the number of iterations in the run where the best log-likelihood is found.

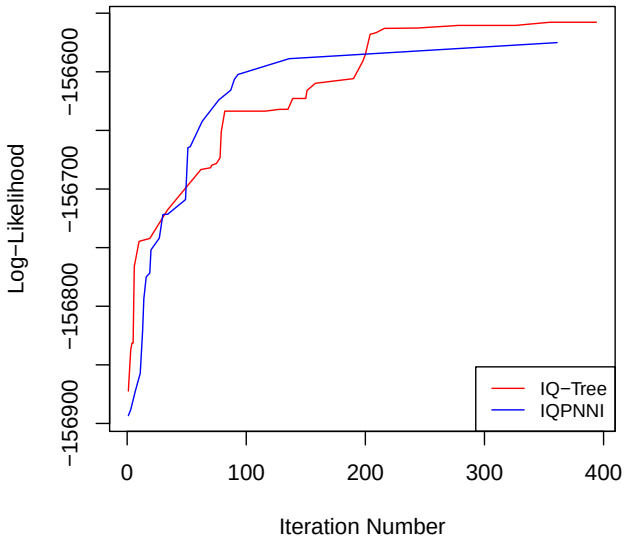

Figure 5.6: Tree log-likelihood improvement by iteration in IQ-Tree and IQPNNI for 218DNA.

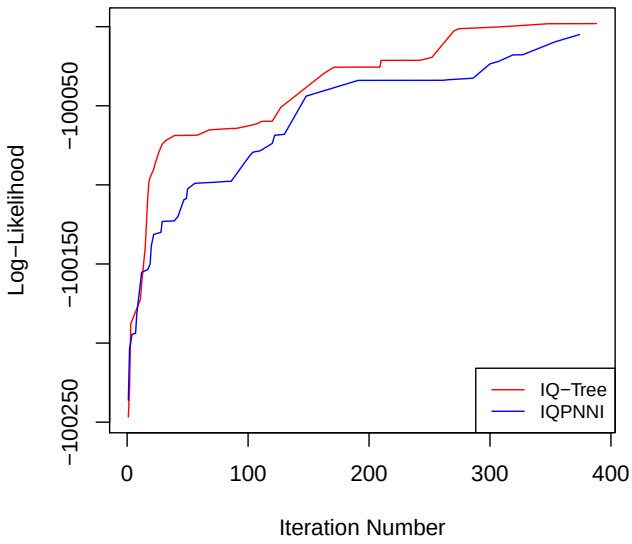

Figure 5.7: Tree log-likelihood improvement by iteration in IQ-Tree and IQPNNI for 500DNA.

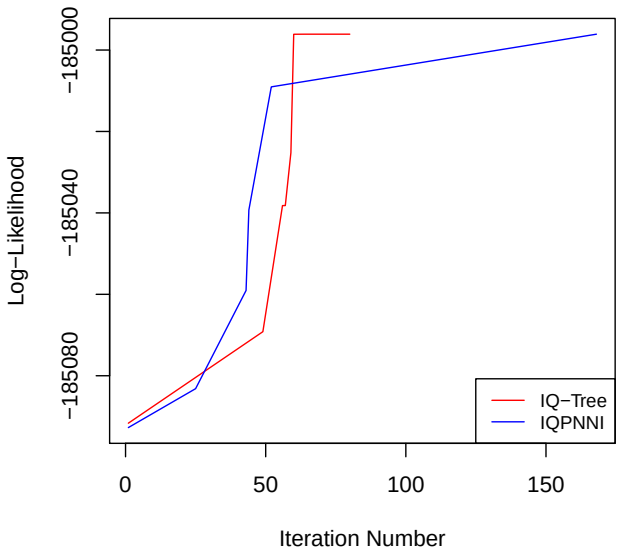

Figure 5.8: Tree log-likelihood improvement by iteration in IQ-Tree and IQPNNI for 74AA.

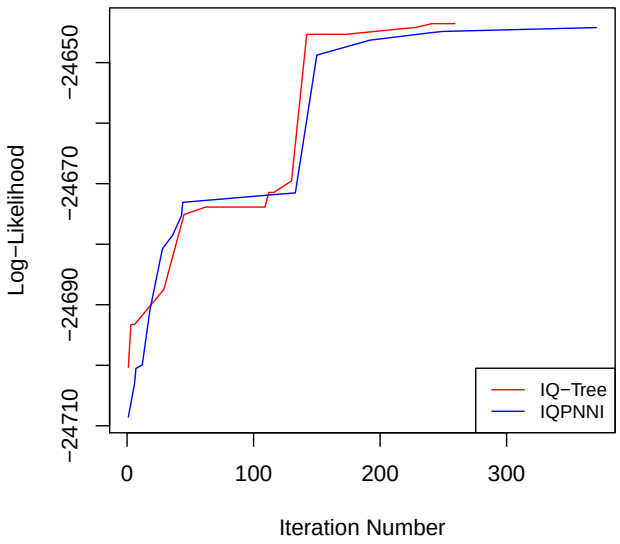

Figure 5.9: Tree log-likelihood improvement by iteration in IQ-Tree and IQPNNI for 144AA.

As we can see from the plots (Figure 5.6, 5.7, 5.8, 5.9) , both IQ-Tree and IQPNNI show similar search behavior during the whole iterations. For data with large number of sequences (218DNA, 500DNA) the best tree log-likelihoods were only found in the last iterations in both programs. Whereas, for smaller data (74AA, 144AA) it takes less number of iterations for the programs to find the best tree log-likelihoods. Especially for the data 74AA, we would assume that the global optimal tree has been found.

The plots and the reported runtime show that IQ-Tree works much more efficiently than IQPNNI.

Chapter 6

Conclusions And Future Work

In this work we have presented two essential runtime improvement techniques for the originally proposed IQPNNI algorithm:

ILS with memory: By presenting IQPNNI algorithm within the framework of ILS, we were able to identify the possibility of utilizing the search history to make the algorithm more efficient. In order to devise a good search history, we investigated the behavior of the search by extracting the likelihood improvements made by NNIs within different iterations of the algorithm. The major feature discovered by our analyses is that the likelihood improvement is by and large linear with the number of NNIs. Thus, we proposed a heuristic in which the likelihood improvement by each NNI and the number of NNIs in each iteration are recorded. This allows us to obtain empirical distributions of the two values. The distributions are used to derive the core values for predicting whether the continuation of the ongoing iteration is worthwhile, i.e. whether or not it would lead us to a better tree. According to our results, the heuristic has made the search run about 1.5 times faster.

Vectorization: We noticed that the formula for computing tree likelihood can be rewritten in form of matrix operations. This led us to the idea of vectorizing the likelihood calculation to speed it up. We have considered the challenges concerning the application of the SIMD instructions for vectorization, which to our knowledge can worsen the performance of the program if not implemented

correctly. Hence, we decided to use the EIGEN linear algebra library for this purpose. As shown in our results, the implemented vectorization generally doubles up the computational speed.

Our results showed that IQ-Tree, with the application of the two techniques, runs two to four times faster than IQPNNI while producing phylogenetic trees of at least equal quality.

Future work

After having achieved the primary aim of the current work: speeding up the computation time of IQPPNI, we are now open for the task of improving the solution quality. To this end, we would like to address some further aspects of the ILS framework in the future work.

One of which is the use of adaptive perturbation (3.3.2). At the moment, as in IQPNNI, we use a fixed scheme for the perturbation method, in which a proportion of the leaves are first deleted and then reinserted into the tree by IQP. This proportion is predefined by the number of sequences in the input alignment and is kept unchanged throughout the search. A drawback of this strategy is that when the search gets stuck in a local optimum, a weak perturbation is sometimes not enough to move the search out of it. Hence, an adaptive perturbation would be helpful in such case. However, one should consider here the trade-off between speed and solution quality, since increasing the perturbation strength will lead to the increased computation time in the local search.

The search would probably benefit from the inclusion of a *tabu list* of deleted leaves in the perturbation step. This works as follows. Every time a leaf is deleted from the tree, it is marked as "tabu" (Glover and Laguna, 1998). The more frequently a leaf is marked as "tabu", the less likely it will be selected for deletion in the next iteration. In addition, we will also consider the possibility of having a more "relaxed" acceptance criterion, in which search diversification is also employed.

Last but not least, it is important for us to be selective in choosing the appropriate modifications so that they will work in harmony.

Bibliography

Addario-Berry, L., Chor, B., Hallett, M., Lagergren, J., Panconesi, A. and Wareham, T. (2003) Ancestral maximum likelihood of evolutionary trees is hard. *Algorithms in Bioinformatics*, pp. 202–215.

Barton, D. E. B. (2007) *Evolution*. CSHL Press.

Battiti, R., Brunato, M. and Mascia, F. (2008) *Reactive search and intelligent optimization*. Springer Verlag.

Benson, D. A., Karsch-Mizrachi, I., Lipman, D. J., Ostell, J. and Wheeler, D. L. (2005) GenBank. *Nucl. Acids Res.*, **33**, D34–D38.

Berkeley (2010a) Building the tree. `http://evolution.berkeley.edu/evolibrary/article/evo_08`.

Berkeley (2010b) Relevance of evolution. `http://evolution.berkeley.edu/evolibrary/article/0_0_0/medicine_01`.

Bertram, J. (2000) The molecular biology of cancer. *Molecular aspects of Medicine*, **21**, 167–223.

Brent, R. (2002) *Algorithms for minimization without derivatives*. Dover Pubns.

Bryant D., G. N. P. M. (2005) Likelihood calculation in molecular phylogenetics. In Gascuel, O. (ed.), *Mathematics of Evolution and Phylogeny*, 1st edition, pp. 33–58, Oxford University Press, USA.

Darwin, C. (1859) On the origin of species by means of natural selection, or the preservation of favoured races in the struggle for life. *New York: D. Appleton*.

Day, W. and Sankoff, D. (1986) Computational complexity of inferring phylogenies by compatibility. *Systematic Zoology*, **35**, 224–229.

Dayhoff, M. and Schwartz, R. (1978) A model of evolutionary change in proteins. In *In Atlas of protein sequence and structure*, Citeseer.

Dobzhansky, T. (1973) Nothing in biology makes sense except in the light of evolution. *The American Biology Teacher*, **35**, 125–129.

Durbin, R., Eddy, S., Krogh, A. and Mitchison, G. (2002) *Biological sequence analysis*. Cambridge university press Cambridge.

Edgar, R. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucleic acids research*, **32**, 1792.

Elias, I. (2006) Settling the intractability of multiple alignment. *Journal of Computational Biology*, **13**, 1323–1339.

Felsenstein, J. (1978) The number of evolutionary trees. *Systematic Biology*, **27**, 27.

Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of molecular evolution*, **17**, 368–376.

Felsenstein, J. (1993) {PHYLIP}: phylogenetic inference package, version 3.5 c.

Felsenstein, J. (1995) PHYLIP (phylogeny inference package), version 3.57 c. *Department of Genetics, University of Washington, Seattle*.

Felsenstein, J. (2004) *Infering Phylogenies*. Sinauer Associates, Sunderland, Massachusetts.

Felsenstein, J. (2008) Treedist – distances between trees. `http://http://evolution.genetics.washington.edu/phylip/doc/treedist.html`.

Flynn, M. (1972) Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, **100**, 948–960.

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

Gascuel, O. (1997) BIONJ: an improved version of the NJ algorithm based on a simple model of sequence data. *Molecular Biology and Evolution*, **14**, 685.

Gill, P., Murray, W. and Wright, M. (1981) Practical optimization.

Glover, F. and Laguna, M. (1998) *Tabu search*. Kluwer Academic Pub.

Guindon, S., Dufayard, J., Lefort, V., Anisimova, M., Hordijk, W. and Gascuel, O. (2010) New algorithms and methods to estimate maximum-likelihood phylogenies: assessing the performance of PhyML 3.0. *Systematic biology*, **59**, 307.

Guindon, S. and Gascuel, O. (2003) A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Systematic biology*, **52**, 696.

Hasegawa, M., Kishino, H. and Yano, T. (1985) Dating of the human-ape splitting by a molecular clock of mitochondrial DNA. *Journal of molecular evolution*, **22**, 160–174.

Hillis, W. D. and Steele, Jr., G. L. (1986) Data parallel algorithms. *Commun. ACM*, **29**, 1170–1183.

Holder, M. and Lewis, P. (2003) Phylogeny estimation: traditional and Bayesian approaches. *Nature reviews genetics*, **4**, 275–284.

Johnson, D. and McGeoch, L. (1997) The traveling salesman problem: A case study in local optimization. *Local search in combinatorial optimization*, pp. 215–310.

Jones, D., Taylor, W. and Thornton, J. (1992) The rapid generation of mutation data matrices from protein sequences. *Computer applications in the biosciences: CABIOS*, **8**, 275.

Jukes, T. and Cantor, C. (1969) Evolution of protein molecules. *Mammalian protein metabolism*, **3**, 21–132.

Kimura, M. (1980) A simple method for estimating evolutionary rates of base substitutions through comparative studies of nucleotide sequences. *Journal of molecular evolution*, **16**, 111–120.

Kirkpatrick, S. (1984) Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, **34**, 975–986.

Kulikova, T., Akhtar, R., Aldebert, P., Althorpe, N., Andersson, M., Baldwin, A., Bates, K., Bhattacharyya, S., Bower, L., Browne, P. *et al.* (2006) EMBL nucleotide sequence database in 2006. *Nucleic acids research*, **35**, D16.

Lemey, P., Salemi, M., Vandamme, A. and Corporation, E. (2009) *The phylogenetic handbook: a practical approach to phylogenetic analysis and hypothesis testing*. Cambridge University Press.

Lemmon, A. and Milinkovitch, M. (2002) The metapopulation genetic algorithm: An efficient solution for the problem of large phylogeny estimation. *Proceedings of the National Academy of Sciences of the United States of America*, **99**, 10516.

Lourenco, H., Martin, O. and Stuetzle, T. (2003) Iterated local search. *Handbook of metaheuristics*, pp. 320–353.

Margush, T. and McMorris, F. (1981) Consensusn-trees. *Bulletin of Mathematical Biology*, **43**, 239–244.

Martin, O., Otto, S. and Felten, E. (1992) Large-step Markov chains for the TSP incorporating local search heuristics. In *Operations Research Letters*, Citeseer.

Minh, B., Klaere, S. and von Haeseler, A. (2006) Phylogenetic diversity within seconds. *Systematic biology*, **55**, 769.

Minh, B. Q. (2005) *Parallel Reconstruction Of Large Maximum Likelihood Phylogenies*. Master's thesis, Albert-Ludwigs University of Freiburg, Germany.

Mount, D. (2004) *Bioinformatics: sequence and genome analysis*. CSHL press.

Needleman, S. and Wunsch, C. (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, **48**, 443–453.

Notredame, C., Higgins, D. and Heringa, J. (2000) T-coffee: a novel method for fast and accurate multiple sequence alignment. *Journal of molecular biology*, **302**, 205–217.

Press, W., Flannery, B., Teukolsky, S., Vetterling, W. *et al.* (2007) *Numerical recipes*, volume 3. Cambridge university press Cambridge.

Rambaut, A. and Grass, N. (1997) Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. *Computer applications in the biosciences: CABIOS*, **13**, 235.

Robinson, L. *et al.* (1981) Comparison of phylogenetic trees. *Mathematical Biosciences*, **53**, 131–147.

Russell, S. and Norvig, P. (2009) *Artificial intelligence: a modern approach*. Prentice hall.

Saitou, N. and Nei, M. (1987) The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular biology and evolution*, **4**, 406.

Schreiber, G. and Martin, O. (1998) Cut size statistics of graph bisection heuristics. *Arxiv preprint cond-mat/9804027*.

Smith, T. and Waterman, M. (1981) Identification of common molecular subsequences. *J. Mol. Bwl*, **147**, 195–197.

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

Stamatakis, A. (2006) RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. *Bioinformatics*, **22**, 2688.

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

Strimmer, K. and von Haeseler, A. (2003) Phylogeny inference based on maximum likelihood methods with tree-puzzle. In Salemi, M. and Vandamme, A.-M. (eds.),

The Phylogentic Handbook, pp. 137–159, Cambridge University Press, Cambridge, UK.

Stuetzle, T. (1999) Local Search Algorithms for Combinatorial Problems: Analysis, Improvements, and New Applications, volume 220 of DISKI. *Infix, Sankt Augustin, Germany*.

Stuetzle, T. and Hoos, H. (1999) Analyzing the run-time behaviour of iterated local search for the TSP. In *III Metaheuristics International Conference*, Citeseer.

Swofford, D. (2002) *PAUP*. Phylogenetic analysis using parsimony (* and other methods). Version 4*. Sinauer Associates, Sunderland, Massachusetts.

Swofford, D. L., Olsen, G. J., Waddell, P. J. and Hillis, D. M. (1996) Phylogeny reconstruction. In Hillis, D. M., Moritz, C. and Mable, B. K. (eds.), *Molecular Systematics*, 2nd edition, pp. 407–514, Sinauer Associates, Sunderland, Massachusetts.

Tavaré, S. (1986) Some probabilistic and statistical problems on the analysis of DNA sequences. *Lec. Math. Life Sci.*, **17**, 57–86.

Thompson, J., Higgins, D. and Gibson, T. (1994) CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic acids research*, **22**, 4673.

Vandamme, A.-M. (2003) Basic concept of molecular evolution. *The phylogenetic handbook: a practical approach to DNA and protein phylogeny*, pp. 1–23.

Vinh, L. and von Haeseler, A. (2004) IQPNNI: Moving fast through tree space and stopping in time. *Molecular Biology and Evolution*, **21**, 1565.

Wang, L. and Jiang, T. (1994) On the complexity of multiple sequence alignment. *Journal of computational biology*, **1**, 337–348.

Whelan, S. (2007) New approaches to phylogenetic tree search and their application to large numbers of protein alignments. *Systematic biology*, **56**, 727.

Whelan, S. and Goldman, N. (2001) A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. *Molecular Biology and Evolution*, **18**, 691.

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

Yang, Z. (2006) *Computational molecular evolution*. Oxford series in ecology and evolution, Oxford University Press.

Zuckerkandl, E. (1987) On the molecular evolutionary clock. *Journal of molecular evolution*, **26**, 34–46.

Zwickl, D. (2006) *Genetic algorithm approaches for the phylogenetic analysis of large biological sequence datasets under the maximum likelihood criterion*. Ph.D. thesis, The University of Texas at Austin.