Parallel Reconstruction of Large Maximum Likelihood Phylogenies

Bui Quang Minh

A Thesis presented for the degree of
Master of Science

Applied Computer Science
Institute for Computer Science
Albert-Ludwigs University of Freiburg
Germany

September 2005

Dedicated to

my beloved parents

Parallel Reconstruction of Large Maximum Likelihood Phylogenies

Bui Quang Minh

Abstract

Interests in understanding phylogenetic relationship among species have been there since Darwin's invention of evolutionary theory (1859). Nevertheless, attempts to mathematically and computationally solve the problem have only been developed since 1960s. Approaches mainly fall into three categories: maximum parsimony, pairwise distance and maximum likelihood. Despite its highly computational expense, maximum likelihood has been observed to frequently achieve better results than others. Therefore, it has received more attention thanks to advances in computer technologies.

Moreover, the recent boom in molecular sequence data has not only helped us recognize the origin of species, but also caused troubles into the efficiency of various methods. Hence, a series of heuristics have been proposed trying to analyze the data in reasonable time. Rencently, the IQPNNI algorithm (Vinh and von Haeseler, 2004), based on maximum likelihood principle, has successfully combined several heuristics together. However, its running time is still not satisfactory. Simultaneously, parallel computing has entered every field of study, which promises to further speed up the methods.

Against this background, this study examined the IQPNNI targeting at optimizing the sequential algorithm. Furthermore, the optimized version was parallelized using message passing, ensuring the scalability on a heterogeneous parallel architecture. Extensive experiments on large datasets have shown significant speedup in both sequential improvements and parallelization. Specifically, the parallelized version pIQPNNI has gained a near linear speedup, while still guaranteeing the accuracy. A saturation effect of the speedup is not apparent due to minimal communication overhead. The runtime reduction and parallel scaling behavior suggest that pIQPNNI is well fitted to reconstruct large phylogenies.

Declaration

The work in this thesis is based on research carried out at the Albert Ludwigs University of Freiburg, Institute for Computer Science, Germany, in cooperation with the Heinrich Heine University of Düsseldorf, Department of Bioinformatics, Germany. No part of this thesis has been submitted elsewhere for any other degree or qualification and it all my own work unless referenced to the contrary in the text.

Acknowledgements

First and foremost, I would like to thank my supervisor, Professor Arndt von Haeseler, for his constant guidance and advice. His brilliant knowledge and invaluable suggestions have always inspired me, from which I have learnt a lot.

Second, I want to thank Professor Susanne Albers for providing me an opportunity working in the field of bioinformatics as well as for her kindness and hospitality.

Third, I would also like to thank Lê Sỹ Vinh for introducing me to Arndt von Haeseler and for his help getting used to the IQPNNI source code. Frequent discussions with him and Heiko A. Schmidt have clarified many things in evolution and parallel computing. In particular, I would appreciate very much Schmidt's careful reading of the manuscript. I also want to thank him and Dorit Liebers for kindly providing the mito74 dataset.

Friendly environment with all other members of bioinformatics department, especially Tanja Gesell, Thomas Schlegel, Ingo Ebersberger, Ricardo de Matos Simoes and Claudia Kiometzis, has made my time full of pleasure and enjoyment. I want to thank Lutz Voigt for setting up the Linux cluster for testing purpose.

Financial support by Konrad Adenauer Stiftung (KAS) is gratefully acknowledged. I would like to thank Berthold Gees for his kind contacts and helps during seminars organized by KAS. Monthly meetings with other fellows have made joy and delight during my free time.

This thesis would not have been possible without my beloved parents, who were always there beside me and encouraging me throughout every moment. They are a reminder that one would never satisfy with his current knowledge. I am also indebted to Nguyễn Kim Ngân for her continuous support in the midst of pressures.

Finally, to all of my friends I express my gratitude.

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

The work presented in this thesis deals with the reconstruction of evolutionary trees, one of the main objectives in bioinformatics. Particularly, we focus on an improved and parallelized implementation of an efficient heuristic method IQPNNI as recently proposed by Vinh and von Haeseler (2004).

In this chapter we firstly start with the motivation of phylogenetic analysis in section 1.1. Secondly, the organization of biological data and main approaches, in which maximum likelihood has played an important role, are described in section 1.2. Next, an overview on parallel computing is given in section 1.3. Finally, we summarize main contributions of this work in section 1.4 and outline the overall structure of the remainder of the thesis in section 1.5.

1.1 Motivation

Charles Darwin (1859) said in his famous *evolutionary theory* that all species have evolved from ancestors under the pressure of *natural selection*. The structure of evolution can be formed as *evolutionary trees*. Such trees are also called *phylogenetic trees* or *phylogenies*. Figure 1.1 illustrates an example phylogeny where Chimpanzee is more closely related to Human than Gorilla and Rhesus.

The interest in understanding this relationship among species has been there for 140 years since Darwin's foundation. However, attempts to mathematically and computationally solve the problem have only been developed since 40 years

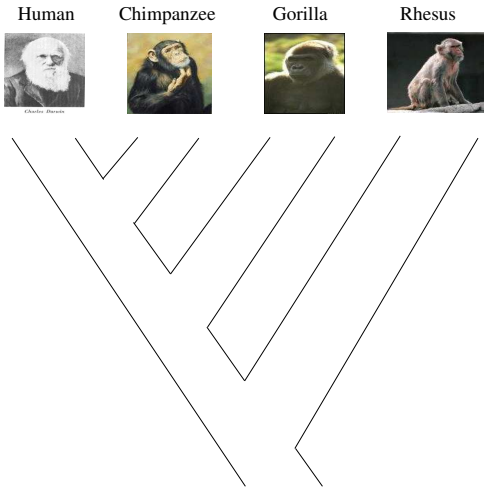

Figure 1.1: An example phylogeny.

(Felsenstein, 2004). The problem can be informally stated as: given information of several contemporary species (presenting as leaves or taxa in the tree), we seek the answers for the following questions:

1. What is the shape of the phylogeny?
2. Who was the most recent common ancestor?
3. How long did a parent species need to evolve into a child?
4. How does an unknown ancestor look like?

Question 1 has been of tremendous interest as obvious fromnumerous literatures (Felsenstein, 2004). The question is also known as *tree topology inference* problem. This task has been, however, well known as being a very difficult problem (Felsenstein, 1978; Foulds and Graham, 1982; Graham and Foulds, 1982; Day and Sankoff, 1986; Day, 1987; Chor and Tuller, 2005). Question 2, also called *ancestral reconstruction*, involves the problem where to place the root of the tree, which is proven to be NP-hard (Addario-Berry *et al.*, 2004). This implies that a polynomial time algorithm to the problems is unlikely.

Question 3 and 4 can be answered if the first two problems are solved. The third question is concerned with determining branch lengths of the tree, or the evolutionary time a species giving rise to its child. We will come back to this in Chapter 3.

Types of Biological Data

Come back to the prerequisites, in which we need information of the taxa. The first kind of data having been analyzed is the morphological characteristics of species (Hillis and Wiens, 2000). They are, for example, eyes colors, face figures or skeleton shapes. Such features can be observed by eyes and easily attained from a wide range of species. It is, however, difficult to differentiate among characteristics, and the number of common morphologies among species is restricted. Hence, they seem not to provide enough information for inferring phylogenetic trees.

Nowadays it is widely accepted to perform analysis on molecular data (Swofford *et al.*, 1996; Felsenstein, 2004), where each species is encoded by a *genome* sequence. A genome consists of several *genes* defining particular functions or properties of species. Two kinds of generally used genetic sequences are *nucleotide* and *amino-acid* data. Nucleotides, the building block for *DNA* sequences, exist in **four** different forms: Adenine, Cytosine, Guanine and Thymine. For *RNA* sequences, Thymine is replaced by Uracil. Amino-acids presented in *protein* sequences consist of twenty

Table 1.1: Twenty amino-acids.

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

different states (Table 1.1). We will only dealing with molecular sequences.

Thanks to recent advance in sequencing technologies, a large amount of genes and genomes have been decoded. The number of sequenced genes and other regions of the genome grows exponentially. They can be accessed from public databases such as GenBank or EMBL (Benson *et al.*, 2005; Kanz *et al.*, 2005). This trend concurrently calls for methods to efficiently reconstruct phylogenetic trees from large datasets.

1.2 Molecular Phylogenetics

In the following we briefly describe methodologies for DNA sequences, in which each character has four different states A, C, G or T. It can be, however, easily adapted to protein sequences by changing the number of states to twenty.

Sequences are called *homologous* if they have descended from a recent common ancestral sequence, such that the sequence information has retained enough similarity for using in phylogenetic analysis (Swofford *et al.*, 1996; Vandamme, 2003). From now on, we are only interested in homologous sequences, and the term "sequence" is referred to the homologous one.

For the sake of simplicity, the tree is restricted to be *bifurcating*, which presumes that any ancestor can give rise to only two separate lineages during the course of evolution (Vandamme, 2003). Moreover, due to the reversibility of the evolutionary model (see Section 1.2.3), the place of the root is irrelevant to the computation, and the *unrooted* tree is usually considered. The remainder of the thesis only deals with bifurcating unrooted trees.

Heuristic Methods

Despite those assumptions, the tree space still grows exponentially: the number of bifurcating unrooted trees is $\prod_{i=3}^{n}(2i-5)$, where n is the number of leaves (Felsenstein, 1978). To avoid an exhaustive search, a number of heuristics have been proposed trying to analyze data within an acceptable amount of time. They have been implemented in many softwares such as PAUP* package (Swofford *et al.*, 1996), fastDNAml (Olsen *et al.*, 1994), PHYML (Guindon and Gascuel, 2003) or AxML

(Stamatakis and Ludwig, 2004).

A commonly used strategy is to combine several heuristics together. IQPNNI (Vinh and von Haeseler, 2004) has been shown to successfully apply such technique. The IQPNNI algorithm will be discussed in details in Chapter 2.

1.2.1 Sequence Alignment

In the evolutionary process, an organism replicates its genome. The copy is inherited into children. This process is not always error-free caused by mutagens like certain chemicals or radiation. Such mutations are therefore accumulated as evidence of evolutionary development. Point mutations, which affects a single character of the DNA, are usually taken into account (Vandamme, 2003) and have three different forms:

- **Substitutions**: replace a character by another.
- **Insertions**: insert one or more characters into the sequence.
- **Deletions**: delete one or more characters from the sequence, indicated by *gap* character '-'.

Sequences are typically presented in a *multiple sequence alignment* (MSA). Figure 1.2 illustrates an example MSA with Human, Chimpanzee, Gorilla and Rhesus. In general, an MSA is defined as a matrix, in which sequences are *aligned* to make

Figure 1.2: Multiple sequence alignment.

	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	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

homologous characters be on the same column (or site) (Waterman, 1995). An example of substitution is at site 6 between Human and Chimpanzee, where 'C' is replaced by 'G' or vice versa. Similarly, site 2 indicates the deletion of 'T' from Chimpanzee to Gorilla, or the insertion of 'T' into Gorilla. Normally, the ancestral sequence is unknown, and therefore insertion or deletion is not clearly distinguishable. Hence, they are referred to *indel* events.

The MSA on n sequences can be constructed in time $O(m^n 2^n)$ and memory $O(m^n)$ applying dynamic programming, where m is the number of sites (Waterman, 1995). It is practically infeasible for a large number of sequences. Thus, approximation methods have normally been employed such as CLUSTALW (Thompson *et al.*, 1994), DIALIGN (Morgenstern, 1999), T-COFFEE (Notredame *et al.*, 2000) or MUSCLE (Edgar, 2004).

1.2.2 Approaches

A variety of methods have been proposed which fall into two categories: *character-based* and *distance-based*. The former always takes into account the content of sequences during the tree reconstruction stage. In contrast, the latter first measures the dissimilarity between all pairs of sequences and subsequently estimates the phylogeny only based on that pairwise distance matrix.

Among character-based methods are *maximum parsimony* (MP) and *maximum likelihood* (ML) two main frameworks, which were first introduced into the field by Edwards and Cavalli-Sforza (1964). MP tries to find a topology that minimizes the number of character changes along the tree, and was proven to be NP-complete (Foulds and Graham, 1982; Graham and Foulds, 1982). ML differs from MP in that it evaluates the likelihood of the tree, defined as the probability of observing the data given the tree. It then attempts to acquire a tree which maximizes this likelihood. ML was also proven to be NP-hard (Chor and Tuller, 2005).

Almost all approaches utilize an objective function (e.g. parsimony or likelihood) to find out a tree which optimizes that function. Among distance-based methods, the famous Neighbor Joining (NJ) algorithm also aims at obtaining a tree which minimizes the sum of all branch lengths (Saitou and Nei, 1987). The widely-used

BIONJ (Gascuel, 1997) is an improved version of NJ which can build a tree in reasonable time. BIONJ serves as the starting point for IQPNNI to further improve the tree (Vinh and von Haeseler, 2004).

In this thesis we mainly discuss the ML, which has been shown to often perform better than other methods in practice (Tateno *et al.*, 1994; Spencer *et al.*, 2005). For that reason, ML has gradually achieved more attention despite its highly computational cost. It is based on a number of assumptions which will be described in the next two sections.

1.2.3 Model of Evolution

To conveniently mathematically model the evolutionary process of nucleotides, i.e., the probability of change (or substitution) from a nucleotide x to a nucleotide y over time t, we have to make several assumptions:

- *Markov process*: The rate of change from x to y is independent of the history of x.
- *Time-continuous*: The substitution can take place at any time point.
- *Time-homogeneous*: The rates of substitution remain constant over time.
- *Stationary*: The frequencies of all nucleotides $(\pi_A, \pi_C, \pi_G, \pi_T)$ are at equilibrium during the course of evolution.
- *Time-reversible*: The substitution rate from x to y during t is the same as the rate from y to x during t.

Given those preconditions, we can now define a so-called *instantaneous rate matrix* $\mathbf{Q}$, which is called the general time reversible model (GTR; Tavaré, 1986; Swofford *et al.*, 1996; Strimmer and von Haeseler, 2003; Felsenstein, 2004):

$$\mathbf{Q} = \begin{pmatrix} - & \alpha\pi_C & \beta\pi_G & \gamma\pi_T \\ \alpha\pi_A & - & \delta\pi_G & \varepsilon\pi_T \\ \beta\pi_A & \delta\pi_C & - & \eta\pi_T \\ \gamma\pi_A & \varepsilon\pi_C & \eta\pi_G & - \end{pmatrix}, \tag{1.1}$$

where the diagonal elements of $\mathbf{Q}$ are specified such that the sum of every row is equal to zero. Rows and columns are ordered as A, C, G and T. For example, the instantaneous rate of substitution from A to G is $\beta\pi_G$.

From Equation 1.1 and homogeneous assumption, the substitution probabilities from any nucleotide to any other during time t can be formulated as a transition probability matrix $\mathbf{P}(t)$:

$$\mathbf{P}(t) = e^{\mathbf{Q}t} \tag{1.2}$$

Matrix $\mathbf{Q}$ can be diagonalized and after further mathematical inference we end up with:

$$\mathbf{P}_{xy}(t) = \sum_{k\in\{A,C,G,T\}} e^{\lambda_k t}\mathbf{U}_{kx}\mathbf{U}^{-1}_{yk}, \text{ with } x, y \in \{A, C, G, T\}, \tag{1.3}$$

where λ_k are the eigenvalues of $\mathbf{Q}$, $\mathbf{U}$ is the matrix with the corresponding eigenvectors, and $\mathbf{U}^{-1}$ is the inverse matrix of $\mathbf{U}$ (Strimmer and von Haeseler, 2003).

Note that matrix $\mathbf{Q}$ is typically standardized such that the total number of substitutions per unit time is one, i.e.,

$$\begin{aligned}&\sum_{x\neq y} \pi_x Q_{xy} = 1, \text{ or}\\ &2\alpha\pi_A\pi_C + 2\beta\pi_A\pi_G + 2\gamma\pi_A\pi_T + 2\delta\pi_C\pi_G + 2\varepsilon\pi_C\pi_T + 2\eta\pi_G\pi_T = 1\end{aligned} \tag{1.4}$$

The GTR model comprises ten parameters $\alpha, \beta, \gamma, \delta, \varepsilon, \eta, \pi_A, \pi_C, \pi_G, \pi_T$, but only eight free parameters because of two constraints: 1.4 and $\pi_A + \pi_C + \pi_G + \pi_T = 1$. In practice, people often use parameter-less models, such as the HKY85 model (Hasegawa *et al.*, 1985):

$$\mathbf{Q}^{HKY} = \begin{pmatrix} - & \pi_C & \kappa\pi_G & \pi_T \\ \pi_A & - & \pi_G & \kappa\pi_T \\ \kappa\pi_A & \pi_C & - & \pi_T \\ \pi_A & \kappa\pi_C & \pi_G & - \end{pmatrix}, \tag{1.5}$$

where κ is called transition/transversion ratio. HKY85 has four free parameters.

1.2.4 Maximum Likelihood

Although first introduced into phylogenetic analysis by Edwards and Cavalli-Sforza (1964), ML was only made computationally applicable by Felsenstein (1981). In the

probabilistic theory, ML refers to the principle where given some data **D**, we try to make some hypothesis which best explains the data. To this end, we search for the the hypothesis **H** which maximizes its likelihood on the data, which is defined as:

$$L(\mathbf{H}) = P(\mathbf{D}|\mathbf{H}), \tag{1.6}$$

or the conditional probability of the data **D** given hypothesis **H**.

In phylogenetic framework, the data **D** is the sequence alignment and the hypothesis **H** consists of a tree **T** (including topology and branch lengths) and a model of evolution **M**.

Likelihood Calculation

Let the alignment D of n sequences be $(D_1, D_2, \ldots, D_m)$, where m is the number of sites, D_s represents the character states of all sequences at site s, i.e., the column s of the matrix D. For simplicity, we also assume:

- Each site evolves independently.
- Each site also evolves according to the same model M.
- Each lineage (branch) evolves independently.

From the first two assumptions, Equation 1.6 becomes:

$$L(T, M) = P(D_1, D_2, \ldots, D_m|T, M) = \prod_{s=1}^{m} P(D_s|T, M). \tag{1.7}$$

Normally $P(D_s|T, M)$ is very close to zero, and to get rid of numerical inaccuracies, we take the logarithm of the likelihood function:

$$\log L(T, M) = \sum_{s=1}^{m} \log P(D_s|T, M). \tag{1.8}$$

To illustrate how to compute the term $P(D_s|T, M)$, we return to the example in Figure 1.1 where the tree contains four leaves and three internal nodes. The same tree topology is depicted in Figure 1.3, which additionally specifies the character states of site s at every node $(x_1, \ldots, x_7)$ and the branch lengths $(t_1, \ldots, t_6)$. Note that x_1, x_2, x_3 and x_4 are taken from the alignment, i.e., constitute the data, while

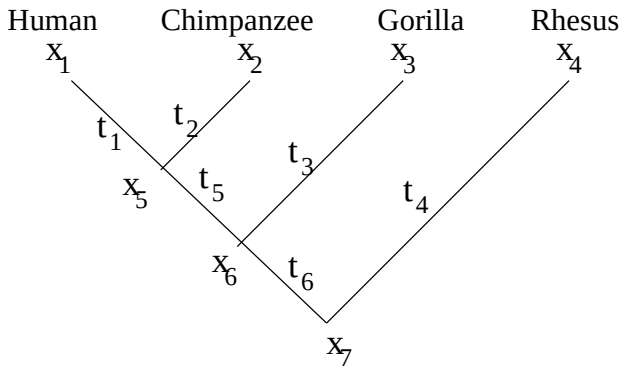

Figure 1.3: Compute likelihood of the example tree.

x_5, x_6 and x_7 are unknown. Together with the third assumption we can multiply the probability of the event occurred along each lineage and it follows:

$$\begin{aligned} P(x_1, \ldots, x_7|T, M) &= P(x_7) \\ & \quad P(x_6|x_7, t_6)\ P(x_4|x_7, t_4) \\ & \quad P(x_5|x_6, t_5)\ P(x_3|x_6, t_3) \\ & \quad P(x_1|x_5, t_1)\ P(x_2|x_5, t_2). \end{aligned} \tag{1.9}$$

where $P(x_7)$ is the prior probability of x_7, or the state frequency π_{x_7}, $P(x_i|x_j, t_i)$ is the probability of x_j giving rise to x_i during evolutionary time t_i. Recall to Section 1.2.3, $P(x_i|x_j, t_i)$ is exactly the term $P_{x_j x_i}(t_i)$ of Equation 1.3.

What we want is to calculate $P(D_s|T, M) = P(x_1, \ldots, x_4|T, M)$, which is simply the sum of the left hand side of Equation 1.9:

$$P(x_1, \ldots, x_4|T, M) = \sum_{x_5} \sum_{x_6} \sum_{x_7} P(x_1, \ldots, x_7|T, M). \tag{1.10}$$

Felsenstein's Pruning Algorithm

Applying Equation 1.10 directly is infeasible in practice. Fortunately, if we rewrite it by combining with Equation 1.9, it follows:

$$\begin{aligned} & P(x_1, \ldots, x_4|T, M) = \\ & \sum_{x_7} \pi_{x_7} \left\{ \sum_{x_6} P_{x_7 x_6}(t_6) \left[\sum_{x_5} P_{x_6 x_5}(t_5) P_{x_5 x_1}(t_1) P_{x_5 x_2}(t_2) \right] P_{x_6 x_3}(t_3) \right\} P_{x_7 x_4}(t_4). \end{aligned} \tag{1.11}$$

Equation 1.11 suggests a recursive bottom-up scheme for general case in the following. To compute the likelihood $P(D_s|T, M)$ at site s, for each node i we define a vector of the so-called *partial likelihoods* $(L_s^i(A), L_s^i(C), L_s^i(G), L_s^i(T))$ as follows:

- If i is a leaf:

$$L_s^i(x) = \begin{cases} 1, & \text{if } x = x_i, \\ 0, & \text{otherwise.} \end{cases}, \ x \in \{A, C, G, T\}, \tag{1.12}$$

  where x_i is the character of sequence i at site s.

- If i is an inner node with two descendants j and k with branch lengths t_j and t_k of (i, j) and (i, k) respectively, then

$$L_s^i(x) = \left[\sum_{y\in\{A,C,G,T\}} P_{xy}(t_j)L_s^j(y)\right]\left[\sum_{y\in\{A,C,G,T\}} P_{xy}(t_k)L_s^k(y)\right], \ x \in \{A, C, G, T\}. \tag{1.13}$$

Let r be the root node, then the likelihood at site s

$$P(D_s|T, M) = \sum_{x\in\{A,C,G,T\}} \pi_x L_s^r(x). \tag{1.14}$$

Note that although having to place a root in this recursive procedure, the resulting likelihood will still remain the same if we move the root to somewhere else due to the time-reversible property of the model. This is called pruning algorithm as proposed by Felsenstein (1981).

Despite the fact that we have made quite a number of assumptions as well as pruning algorithm to simplify the calculation of the likelihood function, ML is still computationally expensive in comparison to MP and distance approaches. Therefore, an ML-based method often dedicates most of its running time only to evaluate the likelihood of the tree.

1.3 Parallel Computing

Facing such computational expenses, efficient heuristic approaches are still overwhelmed once the amount of data becomes larger with thousands of sequences.

Such obstacles can be overcome by exploiting the fast growing parallel platforms, at which a number of processors can work concurrently together towards the goal. The application assigns its tasks onto different processors, which promises to speed it up further. The *speedup* factor on a homogeneous parallel platform, i.e. all CPUs run at the same speed and have similar configuration, is measured by:

$$\text{Speedup on } \mathbf{p} \text{ CPUs } = \frac{\text{sequential runtime}}{\text{runtime on } \mathbf{p} \text{ CPUs}} \tag{1.15}$$

For an overview of parallelization employed to bioinformatic applications, refer to Trelles (2001).

1.3.1 Type of Parallel Platforms

Nowadays, it is commonly admitted to classify parallel systems into three types based on the internal memory architecture: *shared-memory*, *distributed-memory* and *hybrid* systems (Hwang and Xu, 1998). In shared-memory architecture, any process running on any CPU has a direct access to the whole memory of the system, i.e., different processes share the same physical memory addresses. Distributed-memory systems, on the contrary, do not have this feature, since every process only has access its own local memory and communicates with others through an interconnected network.

Shared memory model brings several benefits. Firstly, the unique memory address space made easy the design of parallel softwares. Secondly, the speed at which a memory byte is to be accessed is equal among processes, which minimizes the communication (Trelles, 2001). However, the costs and complications in designing and setting up a shared memory computer nearly prohibit its extension to more than 64 processors. Distributed memory, on the other hand, exhibits a good scalability. It, nevertheless, incurs waiting time for communication between processes during message transmission and synchronization phases. As a result, various means have to be considered to reduce the communication overhead.

To prevail over such disadvantages, shared- and distributed-memory are regularly combined to establish a hybrid system: several shared-memory nodes are connected via a high throughput network. This makes possible installing high performance

computing systems with thousands of processors.

Symmetric Multi-Processing (SMP) is well known as the design of shared-memory: multiple CPUs reside on the same cabinet. When necessary, additional CPUs can be inserted into the computer. Currently, SMPs typically support 2 to 64 processors. Hence, several SMPs are often clustered by a high bandwidth network to create a hybrid system.

An easily inducted and very promising parallel platforms are the so-called clusters of workstations (COWs), where several microcomputers are connected through a local area network. COWs are the most abundant parallel architecture in biological research (Trelles, 2001). Such a cluster typically exists at any work place owing to its cheap configuration and maintenance. Moreover, the advance in gigahertz microchips as well as gigabit ethernet has made it sufficient to serve high performance computing.

1.3.2 Programming Models

For distributed memory model, widely used Message Passing Interface (MPI) was first specified in 1994 as an industrial standard for parallel computing using explicit message passing (Snir *et al.*, 1998; Gropp *et al.*, 1998). MPI implements on a broad range of message passing mechanisms to optimize for communication speed. It has been supported on almost all parallel platforms like massively parallel processors, SMPs, and COWs. Several MPI implementations are freely available such as LAM-MPI or MPICH (see the web page http://www.lam-mpi.org/mpi/implementations/ for a list of currently available libraries).

For shared memory parallelism, OpenMP has emerged as an industrial standard (Dagum and Menon, 1998). It makes use of multi-threading to distribute the computation onto different CPUs implicitly. Programmers only have to take care of some simple directives and thread safety in their source code, which will be ignored by a non parallel compiler. The effort in developing a parallelized program in OpenMP is thus easier than in MPI, however, it is only restricted to SMPs.

At the moment, both MPI and OpenMP are supported in Fortran, C and C++. Several vendors have integrated a feature to correctly compile hybrid programs,

which use both standards. Hence, they offer the great opportunity to develop software running efficiently on large scale hybrid systems like clusters of SMPs.

In evolutionary research, parallel computing has been successfully applied through a range of parallelized phylogenetic tools such as fastDNAml (Olsen *et al.*, 1994), TREE-PUZZLE (Schmidt *et al.*, 2002), AxML (Stamatakis and Ludwig, 2004), MrBayes (Altekar *et al.*, 2004) and DPRml (Keane *et al.*, 2005). Most of these applications are based on MPI or OpenMP (except that DPRml uses its own Java distributed library).

1.4 Contribution

We introduce an algorithmically improved version of the IQPNNI (Vinh and von Haeseler, 2004), a recently proposed method to efficiently reconstruct the maximum likelihood tree from large datasets. The optimized version is furthermore parallelized using MPI. We used MPI because it is supported on almost all platforms, thus ensuring the portability among parallel architectures. Another reason is the *coarse-grained* structure of the algorithm, which facilitates an efficient message passing parallelization.

Extensive analyses on a Linux cluster showed substantial speedup for both sequential and parallelized versions, while still guaranteeing the accuracy of the method.

Part of this thesis was published in the following article:

Bui Quang Minh, Le Sy Vinh, Arndt von Haeseler, Heiko A. Schmidt (2005) pIQPNNI: Parallel reconstruction of large maximum likelihood phylogenies. *Bioinformatics*, in press/advance online (DOI: 10.1093/bioinformatics/bti594).

pIQPNNI source code as well as example data are freely available from http://www.bi.uni-duesseldorf.de/software/iqpnni.

1.5 Thesis Structure

The remainder of this thesis is structured as follows:

- Chapter 2 summarizes some related works on which the IQPNNI algorithm is based and the underlying *important quartets* concept as the core of the heuristics. The algorithm is then outlined, and finally its efficiency over other widely used methods is given.

- Chapter 3 firstly investigates the sequential program to determine what parts are relevant to be improved. Secondly, better numerical optimization algorithms, especially Newton-Raphson, are described. Analyses on simulated and real large datasets are then conducted to compare the performance with the original version.

- Chapter 4 present a parallelization of the accelerated version. Tests on the same datasets have been intensively studied to evaluate the speedup as well as the accuracy behavior. It is shown that the parallelized version scales near to the theoretical speedup. Moreover, it helps us hunt for the best trees which can be obtained from this heuristic method.

- Chapter 5 highlights the main conclusions of this study and then discusses some potential future aspects.

Chapter 2

IQPNNI Algorithm

In this chapter we briefly outline the well known quartet puzzling method (Strimmer and von Haeseler, 1996) in Section 2.1 and the recently proposed IQPNNI algorithm (Vinh and von Haeseler, 2004) in the subsequent Sections. The Important Quartet Puzzling (IQP) and Nearest Neighbor Interchange (NNI) are described in section 2.2 and 2.3 respectively. Section 2.4 presents the algorithm. Finally, the performance of IQPNNI against other methods is given in section 2.5.

2.1 Quartet Puzzling

Quartet Tree

A quartet tree is simply defined as a tree of four taxa. Figure 2.1 illustrates all three possible quartet tree topologies with four leaves A, B, C and X, where leaf A can be on the same side with X, B, or C.

Quartet trees serve as the core building block of the QP algorithm, in which the

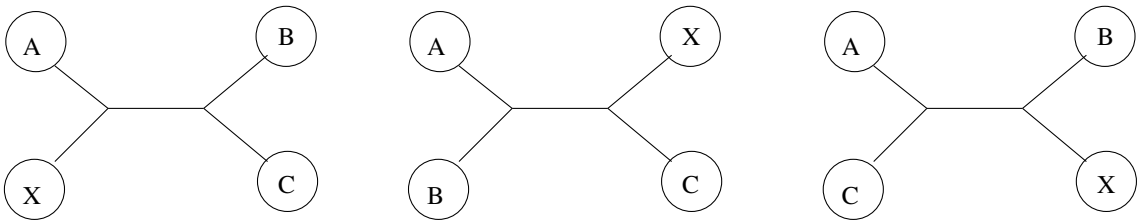

Figure 2.1: Three possible quartet trees denoted as: $T_{AX|BC}$, $T_{AB|XC}$, $T_{AC|BX}$.

overall tree with n leaves is constructed so as to realize as many quartet topologies as possible. The idea behinds this is to reduce the computational effort by considering only $O(n^4)$ quartets, where n is the number of species.

Quartet Puzzling Algorithm

The QP algorithm (Strimmer and von Haeseler, 1996) consists of three steps:

Maximum likelihood step: All $\binom{n}{4}$ possible quartet trees are evaluated by their likelihood value. The optimal quartet topologies among three are taken into account in the subsequent stages.

Puzzling step: At first, an initial tree with any three leaves is built. The tree is then constructed in *stepwise addition* manner, in which taxa are added into the tree by a random order. When inserting a leaf X into the current tree, we consider all quartets containing four leaves A, B, C and X, where A, B, and C are from the current tree. Leaf X is then added to a branch contradicting with least quartets. The puzzling step is repeated several times to search the tree space thoroughly in the hope of avoiding a local optimum. The full tree is called intermediate tree.

Consensus step: From all intermediate trees constructed in the puzzling step, a consensus tree is inferred, which attaches each branch with a support value (How many trees contain the split by this branch).

The final consensus tree is called *quartet puzzling tree*. Users are advised to only trust branches whose support value is greater than 50%.

2.2 Important Quartet Puzzling

Quartet puzzling still disables us from real analysis due to its complexity $O(n^4)$ when n becomes large. IQP (Vinh and von Haeseler, 2004) is a simplified but computationally more efficient version of the QP method by examining only $O(n)$ quartets, the so called *important quartets* (IQs), which in turn reduces the complexity to only

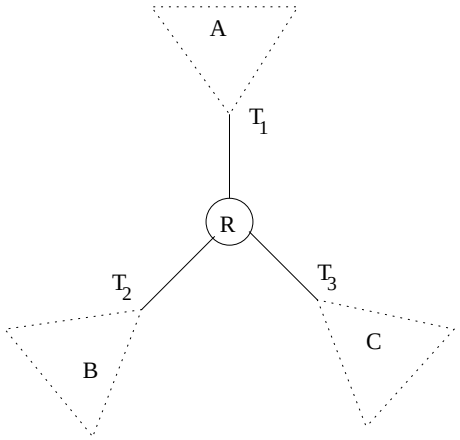

Figure 2.2: Important quartets with respect to a virtual root R breaking the tree into three subtrees T_1, T_2 and T_3. A, B and C are picked up from the k-representative sets of T_1, T_2 and T_3 respectively.

$O(n^2)$. An IQ also contains four leaves A, B, C, and X as in the *puzzling step* (see Section 2.1), but we constrain A, B, C *near enough* to each other. This is defined by the help of a so called *k-representatives* concept (see also Vinh and von Haeseler, 2005): given the current tree rooted at an arbitrary node, the k-representatives of the tree is the set of at most k leaves with smallest *distance* from the root. The distance between two nodes is simply the length of the path connecting them. In case of ties, leaves with the same distance to the root are picked up randomly.

The k-representatives of a rooted tree can be computed in $O(nk)$ time. For an unrooted tree, we can place a fake root at any internal node. This root breaks our tree into three rooted subtrees (Figure 2.2). If A, B, and C are taken from k-representative set of each subtree respectively, we have no more than k^3 possible IQs. Since there are $n-2$ internal nodes, the total number of considered IQs is at most $(n-2)k^3$. In practice, the parameter k is chosen much smaller than n. Hence, IQP has a time complexity of only $O(n^2)$, which enables its application to large data sets.

2.3 Nearest Neighbor Interchange

NNI is the simplest algorithm among local tree rearrangement methods (Felsenstein, 2004). Figure 2.3 illustrates on the left-hand side a topology connecting four subtrees t_1, t_2, t_3, and t_4. Exchanging the position of t_2 with t_3 or t_4 results in two possible topologies as on the right-hand side. All other swaps will end up at one of these three configurations.

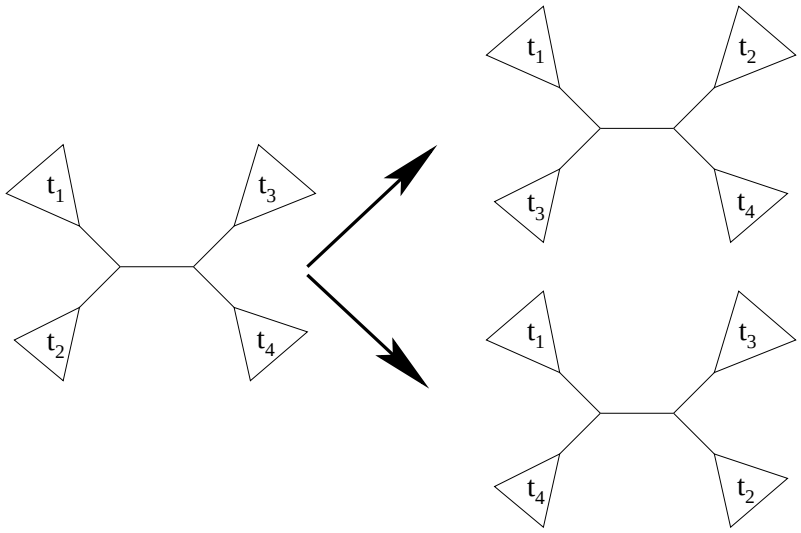

Figure 2.3: NNI operations.

The likelihoods of two subsequent tree topologies are evaluated, and if any of them shows a better likelihood value, the swap is applied. For a tree of n taxa, there exist totally $2(n-3)$ possible NNIs (Felsenstein, 2004). Recently, Guindon and Gascuel (2003) have proposed a fast and efficient NNI strategy, which considers all NNIs together to find out the best collection of non-overlapping NNIs. The tree topology and branch lengths are then simultaneously (locally) optimized. IQPNNI applies the same procedure in its implementation.

2.4 IQPNNI Algorithm Outline

The IQPNNI algorithm (Vinh and von Haeseler, 2004) combines several heuristics together: distance-based method BIONJ (Gascuel, 1997), fast NNI (Guindon and Gascuel, 2003) and its own IQP algorithm. It consists of two major steps: the *initial*

step and the subsequent *optimization step* (OS), where the latter consumes 90% to 99% of the total running time (Figure 2.4).

2.4.1 Initial Step

In the initial step, an initial tree is obtained by applying any fast tree building algorithm. In the original implementation, the distance-based BIONJ method (Gascuel, 1997) is used.

Subsequently, the parameters of the substitution model are estimated, and fast NNI operations (Guindon and Gascuel, 2003) are employed to improve the likelihood of the BIONJ tree topology.

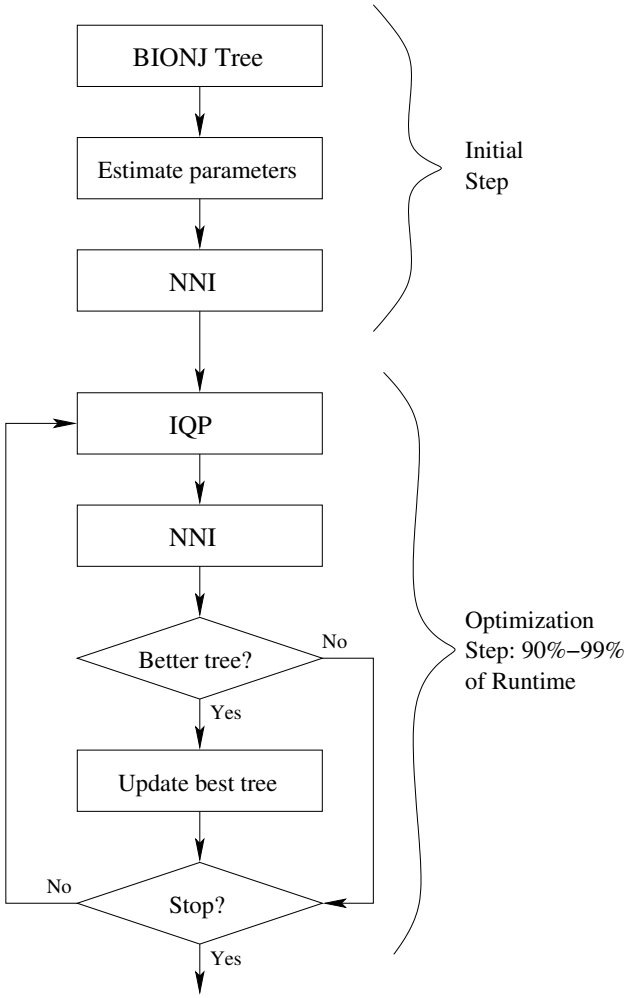

Figure 2.4: Sequential IQPNNI.

2.4.2 Optimization Step

In the OS, a number of leaves are first removed randomly from the tree with a certain probability p_{del}. These leaves are re-inserted in a random order by the IQP algorithm. The resulting tree topology is again optimized by NNI until no further likelihood improvement is achieved. If the likelihood of the resulting tree exceeds the current best likelihood, then the current best tree is replaced by the new one.

The OS is repeated many times to thoroughly search the tree space. Iterations stop if either a user-defined number of repetitions is met, or if the so-called *stopping rule* is fulfilled. The stopping rule is based on a Weibull distribution to decide whether it is likely (with a 95% confidence level) that the continuation of the search will lead to no further improvement.

2.4.3 Parameter Settings

The algorithm has two parameters: the proportion of deleted leaves p_{del} and the number of representatives k. The smaller the p_{del} is, the more preserved the topology of the current tree. The intension of using parameter p_{del} is to improve the current best tree while still somehow reusing its shape information. It is thus important not to set it too small or too large.

If parameter k is set large enough, the IQs will become normal quartets. Hence, when $p_{del} = 1.0$ and $k = n$, the IQP algorithm reduces to the QP algorithm. In real application, however, we often set p_{del} ranging from 0.1 to 0.3 and k between 3 and 5, depending on the number of sequences and sequence length (Vinh and von Haeseler, 2004).

2.5 Performance

IQPNNI were extensively tested to compare the performance with other widely used phylogenetic applications, including PHYML (Guindon and Gascuel, 2003), MetaPIGA (Lemmon and Milinkovitch, 2002), Weighbor (Bruno *et al.*, 2000) and fastDNAml (Olsen *et al.*, 1994). In simulated data, IQPNNI was claimed to outperform other methods in terms of the similarity of the inferred tree with the true tree.

In two real data (Table 2.1), it also achieved higher likelihood tree than PHYML and MetaPIGA. Hence IQPNNI produced more accurate results in both cases.

Table 2.1: IQPNNI vs. PHYML and MetaPIGA (Vinh and von Haeseler, 2004).

Gene	#Sequences	Log-likelihood		
		PHYML	MetaPIGA	IQPNNI
rbcL[1]	500	-100,191	-100,080	**-100,011**
ssu rRNA[2]	218	-156,895	-156,715	**-156,604**
		Runtime in minutes		
rbcL[1]	500	7.5	158.5	**672**
ssu rRNA[2]	218	5.1	74.5	**379**

[1] same as dataset 2 in Table 3.1.

[2] same as dataset 3 in Table 3.1.

IQPNNI consumes, on the contrary, more running time. It took for example 11 hours to analyze the real dataset of 500 sequences, whereas 7.5 minutes with PHYML and 2.6 hours with MetaPIGA (Table 2.1). It would thus be nice to speed up the method so as to make it more feasible solving large datasets. We will see that in the next two chapters.

Chapter 3

Sequential Improvements

There exist different means to accelerate a sequential program. One can optimize the data structures, implement more efficient algorithms, and/or even rewrite some codes in a lower level language. To this end, one should first investigate which parts of the code consume the most running time. This can be solved mentally or physically. In a mental way, one looks at the data structures and algorithms to figure out the time and memory complexity. In a physical way, one can use performance analysis tools to exactly measure the executed time of every single function. We will pursue the latter method in this chapter.

We are first dealing with coding analysis techniques for tuning the program in Section 3.1. Second, a replacement for the most time consuming optimization method is discussed in section 3.2. Next, means to overcome the computational burden of parameter estimation for parameter-rich model is introduced in section 3.3. Finally, section 3.4 gives the performance study of all sequential improvements having been shown so far.

3.1 Implementation Tuning

A programmer can observe by eye which parts of his program take more time to finish during one run. This subjective measurement however will often lead to inexact estimation.

A more widely used statistical method is called *profiling*. It works by automati-

cally adding some extra code in the entry and exit of each function to measure how long and how often this function is executed. It can also record the calling structure to create the so-called *call-graph* of the program. From this profile, the programmer can more exactly and easily analyze his source code to figure out what procedures are more time consuming.

In GNU C/C++ compiler, for example, the profiling feature can be turned on by compiler flags "-pg -fprofile-arcs". Then the program should be executed and afterwards running information will be recorded in some special files. Now using a profiler like "gprof" one can summarize the recorded information into a profile.

We applied these profiling techniques to analyze the 1000 sequence DNA dataset with 300 iterations (Table 3.1). We obtained a call-graph for IQPNNI as illustrated in Figure 3.1. For the sake of clarity, it only retains all core functions and discards less important ones. Interestingly, we found that the IQP procedure took unexpectedly long time (5.8 hours / total 15 hours) compared to the theoretical complexity $O(n^2)$. More careful examination located some inefficient code, which then have

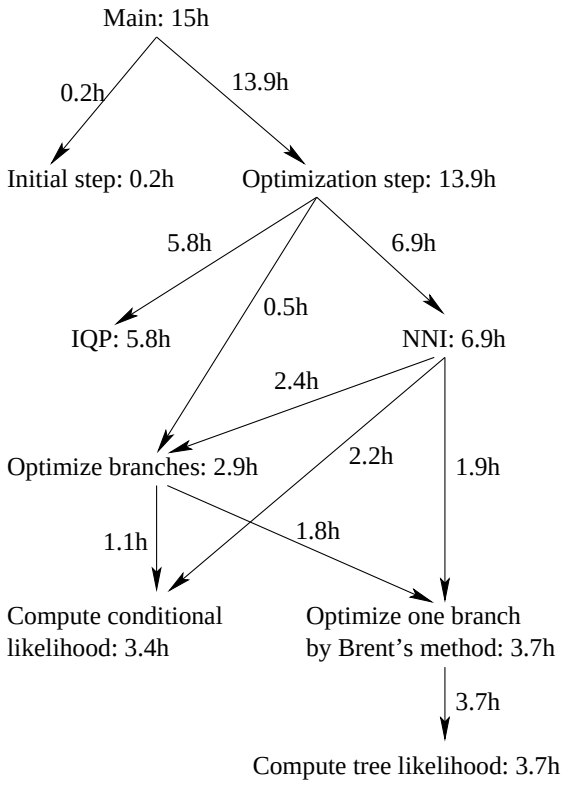

Figure 3.1: Call graph of IQPNNI applied to 1000 sequence dataset in Table 3.1.

been fixed.

After that, the computation of the likelihood consumed almost 95% of the whole running time, in which the application of Brent's method to optimize branch lengths of the tree took more than one half.

3.2 Branch Lengths Optimization

The previous section has revealed that the implementation of Brent's method (1973) to determine the optimal branch lengths consumed the largest fraction of the running time. Newton's method (Press *et al.*, 1992) on the other hand has been shown to be faster by taking advantage of the first and second derivatives which can be efficiently calculated (Yang, 2000). However, just substituting Brent's by Newton's method might be dangerous, since the latter sometimes suffers from a well-known problem of non-convergence (Press *et al.*, 1992). Hence, we applied a slight modification: if Newton's method returns a value, we check whether it is (locally) optimal and re-optimize with Brent's algorithm if necessary.

3.2.1 Newton-Raphson for One Dimension

Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous function, e.g. the likelihood function, and we want to find a point x maximizing it. An important property is that the function gradient at this point is zero, i.e. $f'(x) = 0$. Moreover, according to Taylor theorem, the first derivative of f can be approximated as:

$$f'(x + \delta) \approx f'(x) + f''(x)\delta + \frac{f'''(x)}{2}\delta^2 + \dots \tag{3.1}$$

The term $\frac{f'''(x)}{2}\delta^2 + \dots$ on the right-hand side is normally much smaller than the first part and can therefore be ignored. If we now want to make the left-hand side zero, equation 3.1 follows

$$\delta \approx -\frac{f'(x)}{f''(x)} \tag{3.2}$$

Equation 3.2 defines δ as the step length of Newton's method in its iterative scheme: given an initial point x_0, the next guess is estimated as: $x_{k+1} = x_k -$

Listing 3.1: Newton-Raphson for one-dimensional maximization.

```
x ← initial guess
while δ ≇ 0 do /* not converged */
    Calculate f(x), f'(x), f''(x)
    δ ← f'(x)/ max(|f''(x)|, 1e − 5) /* trick to go towards maximum */
    while f(x + δ) < f(x) ∩ δ ≇ 0 do /* hill climbing */
        δ ← δ/2 /* divide step length by 2 */
    Update x ← x + δ
return x
```

$f'(x_k)/f''(x_k)$. The iteration is stopped if the difference between the current and the previous estimation is not significant anymore.

Newton's method was proven to converge quadratically to an optimum provided that the initial guess is close enough to it (Press *et al.*, 1992). Unfortunately, this is not always the case. Newton's method, moreover, only assures us to reach either a minimum or a maximum. If the current guess lies in the non-convex part of the function, i.e. $f''(x) > 0$, it tends to send us towards a minimum and vice versa. Even in case $f''(x)$ is negative, still Newton's method can be trapped into a *non-convergent cycle* where it sends us back and forth between two points (refer to Press *et al.*, 1992, for more details).

Listing 3.1 outlines the Newton-Raphson algorithm consisting of two loops. The outer loop stops when the step length δ is small enough. Inside the outer loop, the function, first and second derivative values are calculated first and then the step length. Note that the nominator is $\max(|f''(x)|, 1e-5)$ to avoid "division by zero" in case $f''(x)$ is accidentally equal to zero. The reason to take the absolute value of $f''(x)$ is follow:

- If $f''(x) > 0$ then $\delta = f'(x)/f''(x)$, and as seen above, the step length $-\delta$ would send us towards a minimum. This way we can avoid this behavior by moving away from the minimum.

- If $f''(x) < 0$ then $\delta = -f'(x)/f''(x)$ and this step length should bring us close

to the desired maximum.

The inner loop, in which the step length is divided by two until reaching a higher point, acts exactly as *hill climbing* technique. This is to overcome the non-convergence cycle by making sure we always go up hill after each Newton's iteration (Gill *et al.*, 1981).

3.2.2 Derivatives of Likelihood Function

In our case the function f is the log-likelihood of the tree and x is the length of one branch. Branch lengths are optimized one by one and we repeat this procedure until no changes occur. This is the simplest optimization strategy but more efficient than by optimizing branch lengths simultaneously when the tree becomes large. The following equations are according to Yang (2000).

Recall chapter 1 when computing the log-likelihood of the tree by Felsenstein's *pruning algorithm* (1981), the likelihood at a site s with respect to a branch (a, b) of length t can be calculated as:

$$f_s(t) = \sum_{x_a} \sum_{x_b} \pi_{x_a} \mathbf{P}_{x_a x_b}(t) L_a(x_a) L_b(x_b), \tag{3.3}$$

where $L_i(x_i)$ is the conditional probability of observing character states at the leaves in the subtree of node i, given that node i has character state x_i. It is known as *conditional likelihood*. $\mathbf{P}_{x_a y_b}(t)$ is the transition probability from state x_a to state y_b over time t, and from Equation 1.3 we have:

$$\mathbf{P}_{x_a y_b}(t) = \sum_k c_{x_a y_b k} e^{\lambda_k t}, \tag{3.4}$$

where the coefficients $c_{x_a y_b k}$ and λ_k are independent of t. The first and second derivatives of the transition probability can thus be easily computed as:

$$\begin{aligned} \mathbf{P}'_{x_a y_b}(t) &= \sum_k c_{x_a y_b k} \lambda_k e^{\lambda_k t}, \\ \mathbf{P}''_{x_a y_b}(t) &= \sum_k c_{x_a y_b k} \lambda_k^2 e^{\lambda_k t}. \end{aligned} \tag{3.5}$$

Since $L_a(x_a)$ and $L_b(x_b)$ are also independent of t, from equation 3.3 follows:

$$f'_s(t) = \sum_{x_a}\sum_{x_b} \pi_{x_a} \mathbf{P}'_{x_a x_b}(t) L_a(x_a) L_b(x_b),$$
$$f''_s(t) = \sum_{x_a}\sum_{x_b} \pi_{x_a} \mathbf{P}''_{x_a x_b}(t) L_a(x_a) L_b(x_b). \tag{3.6}$$

Since the log-likelihood function $f(t) = \sum_s \log\{f_s(t)\}$, its derivatives are:

$$f'(t) = \sum_s \frac{f'_s(t)}{f_s(t)},$$
$$f''(t) = \sum_s \frac{f_s(t) f''_s(t) - [f'_s(t)]^2}{f_s^2(t)}. \tag{3.7}$$

For branch (a,b), the two terms $L_a(x_a)$ and $L_b(x_b)$ are free of length t and can be pre-computed. Hence, the calculation of $f(t)$, $f'(t)$ and $f''(t)$ can be fast carried out during the optimization process.

3.3 Parameter Estimation

3.3.1 Newton-Raphson for Multiple Dimensions

Equation 3.2 can be easily adapted to multi-dimension with function $f : \mathbb{R}^n \to \mathbb{R}$. The first derivative is now the gradient vector:

$$\nabla f(X) = \begin{pmatrix} \frac{\partial f}{\partial x_1} \\ \vdots \\ \frac{\partial f}{\partial x_n} \end{pmatrix}, \ where\ X = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \tag{3.8}$$

The second derivative is the Hessian matrix containing all partial second derivatives:

$$Hf(X) = \begin{pmatrix} \frac{\partial^2 f}{\partial^2 x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial^2 x_2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial^2 x_n} \end{pmatrix} \tag{3.9}$$

Equation 3.2 becomes:

$$\Delta = -[Hf(X)]^{-1} \nabla f(X) \tag{3.10}$$

The standard algorithm consists of two steps:

1. **Initial**: $X_0 \leftarrow$ first guess.
2. **Iterative**: $X_{k+1} \leftarrow X_k - [Hf(X_k)]^{-1}\nabla f(X_k),\ k \geq 0$, until $|X_{k+1} - X_k|$ is small enough.

3.3.2 Broyden-Fletcher-Goldfarb-Shanno (BFGS) Algorithm

Equation 3.10 involves the calculation of the inverse Hessian matrix which is quite expensive. BFGS algorithm is also called a quasi-Newton method in the sense that it approximates this term. Hence, the method only requires the computation of the gradient vector. Refer to Press *et al.* (1992) for more details about the algorithm.

Unfortunately, it is not easy to compute the gradient of the log-likelihood function with respect to the parameters of substitution model. We can, however, estimate it based on the original definition of the derivative:

$$f'(x) = \lim_{\epsilon \to 0} \frac{f(x+\epsilon)}{\epsilon} \tag{3.11}$$

When ϵ is chosen small enough, the term $\frac{f(x+\epsilon)}{\epsilon}$ will be approximately the first derivative. In practice one would select $\epsilon = 1^{-4}|x|$. This is called the *forward difference* method (Press *et al.*, 1992).

3.3.3 Combined Scheme for Parameter Estimation

The parameter estimations (PEs) are originally implemented by only using Brent's method containing two loops. In the outer loop, parameters are optimized one by one, then entering the inner loop determining branch lengths also step by step until no changes occur. The outer loop is finished if there exist no changes in the estimated parameters (Figure 3.2).

This mechanism is mainly kept in the new implementation with two alterations: parameters are estimated simultaneously by BFGS algorithm and the branch lengths are optimized also one after another but using Newton's method. We do not apply BFGS to optimize model parameters and branch lengths at the same time because such method would become very inefficient when the number of sequences increases (Yang, 2000).

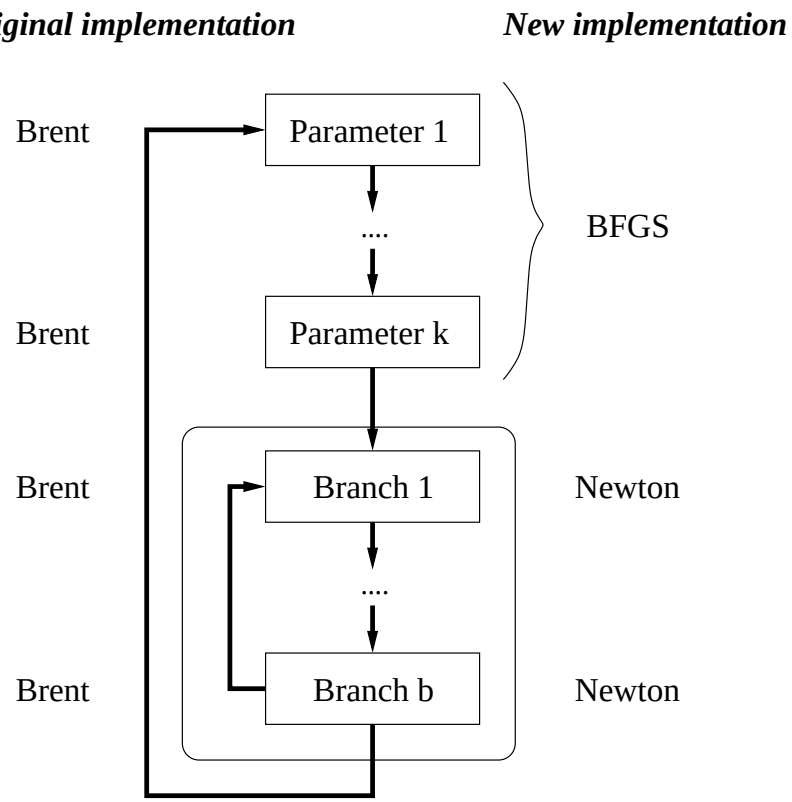

Figure 3.2: Parameter estimations: original vs. new implementation.

3.4 Runtime Analysis

3.4.1 Datasets

The performance of sequential improvements was tested on four datasets (Table 3.1). The first dataset is a simulated DNA data used by Vinh and von Haeseler (2004). Two real well known datasets 2 and 3 were also used by Vinh and von Haeseler (2004); Guindon and Gascuel (2003). The rbcL data was first analyzed by Chase *et al.* (1993) and has been interested in various literatures. The ssu rRNA data was downloaded from the Ribosomal Database Project II http://rdp.cme.msu.edu/. The unpublished protein dataset mito74 was kindly provided from our colleagues H.A. Schmidt and D. Liebers.

As models of substitution, we used the HKY85 (Hasegawa *et al.*, 1985, see also section 1.2.3) for DNA sequences and the WAG (Whelan and Goldman, 2001) for protein sequences. The number of iterations in the optimization step was fixed (see Table 3.1) to avoid runtime fluctuations caused by the stopping rule.

Table 3.1: The datasets used for analysis.

No.	Type	Name	#Seqs	#Sites	#Patterns	#Iterations
1	DNA	sim.	1000	1000	1000	300
2	DNA	rbcL[1]	500	1398	1193	300
3	DNA	ssu rRNA[2]	218	4182	1847	300
4	AA	mito74[3]	74	4013	3038	200

[1] plant plastid gene.

[2] prokaryotic sequences from the small ribosomal subunit.

[3] vertebrate sequences.

3.4.2 Results

We measured and compared the runtime of three versions on a CPU of 2.0GHz:

Original: The original IQPNNI version 2.6. Note that Vinh and von Haeseler (2004) had used an older version which had even longer running time.

Tuning: The tuning version applying techniques described in section 3.1.

T+Newton: The further improved version using Newton's instead of Brent's method to optimize branch lengths of the tree (see Section 3.2).

Table 3.2 shows the runtime and relative speedup among three versions. Applying tuning techniques made the program run from 1.2 to 1.5 times faster, which performs best on the protein data, with respect to speedup, due to the most intensive computation.

Using Newton's method for DNA data we attained a reduction of 33% to 45% in running time. For the protein data, we got a reduction in runtime of (only) 17%. This is due to the fact, that Newton's method suffers from the higher complexity of protein models while computing derivatives. Moreover, Newton's method also did converge slower on the protein data (Table 3.3).

Both improvements resulted in a significant speedup for the sequential program ranging from 1.8 to 2.2. For instance, it needs only 4.7 hours (instead of 9.5 hours

Table 3.2: Sequential runtime and speedup.

Dataset	Runtime			Speedup		
	Original	Tuning	T+Newton	Tuning[1]	T+Newton[2]	Total[3]
1	9h:29	7h:02	4h:43	1.3	1.5	2.0
2	7h:06	5h:32	3h:26	1.4	1.6	2.1
3	5h:27	4h:27	2h:26	1.2	1.8	2.2
4	11h:02	7h:11	6h:14	1.5	1.2	1.8

[1] Tuning vs. Original version.

[2] Tuning+Newton vs. Tuning version.

[3] Tuning+Newton vs. Original version.

Table 3.3: Average steps to convergence per branch.

Dataset	Brent	Newton
1	15.2	1.7
2	15.3	2.1
3	12.0	2.5
4	11.9	2.7

in the original version) to analyze the 1000 sequence dataset. These results suggest that improved IQPNNI version is even more applicable for analyzing large datasets within an acceptable time limit.

BFGS vs. Brent method

To evaluate the performance of the new implementation based on the multi-dimensional BFGS algorithm to estimate the parameters in the initial step, we use the GTR model of substitution for DNA (Tavaré, 1986, see also section 1.2.3). The GTR is a parameter rich model and thus suitable for comparison.

Table 3.4 shows the runtime and the number of iterations the outer loop in figure 3.2 required to converge by both methods (note that the protein dataset

Table 3.4: Parameter estimation: Brent vs. BFGS algorithm.

Dataset	Runtime (sec.)		Iterations	
	Brent	**BFGS**	Brent	**BFGS**
1	521	**128**	29	**3**
2	297	**84**	26	**4**
3	169	**54**	17	**3**

was excluded from the analysis as the GTR model is only for DNA). The new implementation ran three to four times faster than the original implementation using Brent's method. It also took much less iterations for convergence.

Although the runtime shown here is quite short, this new schema for parameter estimations is very advantageous in two ways. First, for higher-dimension parameter-rich models (not just only 4×4 as for DNA), the runtime needed will be longer and the new implementation will be of great benefit. Second, optimizing parameters simultaneously would have a smaller chance of trapping into a local optimum than estimating them one by one.

Chapter 4

Parallelized IQPNNI

Although the algorithmic improvements presented in chapter 3 are substantial, there are even more means to reduce the runtime of the analysis. In the following chapter we present a parallelization of the method using the message passing paradigm. While section 4.1 and 4.2 deal with strategies parallelizing the initial step and the optimization step, respectively, their running time and speedup are analyzed in Section 4.3. Finally, section 4.4 discusses the accuracy of the parallel scheme and will give the best achieved results for two well known real data.

4.1 Parallelization of Initial Step

In the initial step of the IQPNNI algorithm (see Section 2.4.1), an initial tree is obtained using the BIONJ method (Gascuel, 1997) on a pairwise distance matrix.

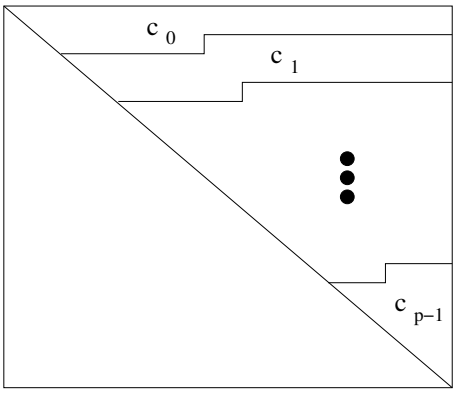

Figure 4.1: Partition work of the upper triangular.

The computation of this matrix using ML takes the longest time compared to a small fraction consumed by the BIONJ algorithm. Parallelizing the ML estimation of the pairwise distances is straightforward.

Since the distance matrix is symmetric, only the upper triangular needs to be computed. The task assignment follows the *static chunking* method (Hagerup, 1997), i.e., if p denotes the number of processors and n is the number of sequences, the half-matrix will be divided into p approximately equal contiguous regions. Each process is responsible for computing $n(n-1)/2p$ pairwise distances belonging to its assigned region (Figure 4.1). Once finished, the results are shared among all processes.

4.2 Parallelization of Optimization Step

The optimization step (OS, Section 2.4.2), even after improving the sequential code, consumes 90% to 99% of the total running time, depending on the number of iterations and the size of the data. Hence, total running time will benefit a lot from an efficient parallelization of the OS.

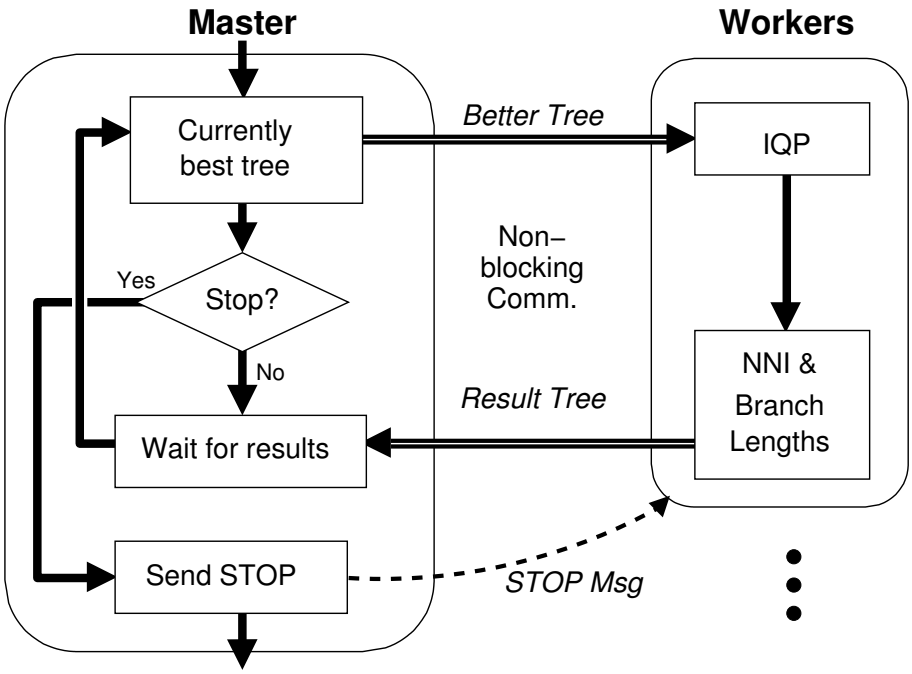

Figure 4.2: Parallelization Scheme of Optimization Step.

Listing 4.1: Master process.

```
Create T_BIONJ tree, send it to workers
Estimate parameters and apply NNI to T_BIONJ
T_master ← T_BIONJ
while not stop do /* reach pre-defined #interations or fulfill stopping rule */
    Wait tree T_intermediate from workers
    if lnL(T_intermediate) > lnL(T_master) then
        T_master ← T_intermediate
        Non−blocking send T_master to other workers
Send STOP signal to all workers
Output T_best
```

Listing 4.2: Worker process.

```
Receive T_BIONJ tree from Master
Estimate parameters and apply NNI to T_BIONJ
T_worker ← T_BIONJ
while not get STOP signal from Master do
    /* computational phase here */
    Create T_intermediate from T_worker by IQP
    Apply NNI to T_intermediate
    if lnL(T_intermediate) > lnL(T_worker) then
        T_worker ← T_intermediate
    /* synchronization phase with Master */
    Non−blocking send T_intermediate to Master
    while Master sent a tree do
        Receive T_master
        if lnL(T_master) > lnL(T_worker) then
            T_worker ← T_master
```

The OS was parallelized using a master/worker scheme (Figure 4.2): A master is coordinating and collecting the results and workers are carrying out the actual computation.

Since the OS acts on the current best tree, iterations in the original version are not independent. To account for this, the sequential scheme was slightly modified: starting from the current best tree, every worker runs its own OS as explained in Chapter 2. When sending back its result, the current best tree is updated by the master, if the returned tree shows a higher likelihood. In such case, the master will broadcast the better tree to all workers by non-blocking communication. This ensures that a worker finishes its current iteration even if a new better tree is sent. This also takes advantage of the time saving by overlapping computation and communication (Snir *et al.*, 1998). After sending its resulting tree and likelihood to the master, the worker checks for a new best tree and starts the next iteration. In addition, the master checks whether the stop condition applies and, if so, sends a stop message to all workers.

Listings 4.1 and 4.2 detail the algorithm, where the master and workers keep their own trees T_{master} and T_{worker} respectively. T_{master} is, at any time, the global best tree found, while T_{worker} remains local and is synchronized with T_{master} after one iteration at workers. The master spends most of its time waiting for some worker and only takes care of synchronization among worker processes. Workers also keep T_{worker} updated, either from its last constructed tree or from the master. Note that each worker might have to check several trees received from the master, due to the fact that more than one better tree could be sent during the worker's computational OS phase.

4.3 Speedup Analysis

The performance of parallelization was experimented on the same datasets with the same parameters as in Section 3.4. We had to refrain to relative speedup because a best known sequential implementation does not exist (see Equation 1.15). Each analysis is repeated 10 times on a homogeneous Linux cluster of 15 nodes with 2

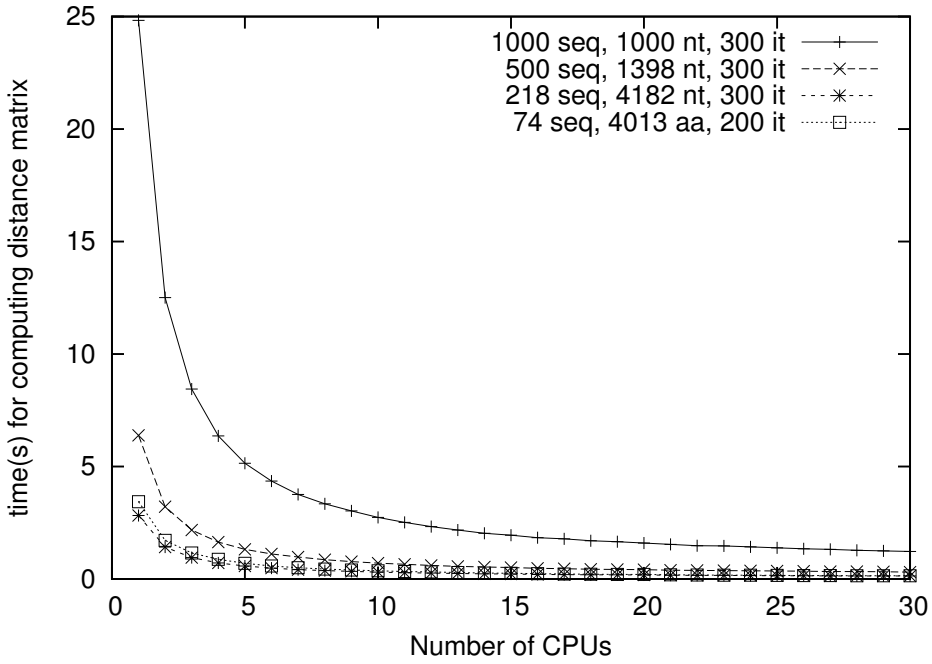

Figure 4.3: Running time for computing distance matrix.

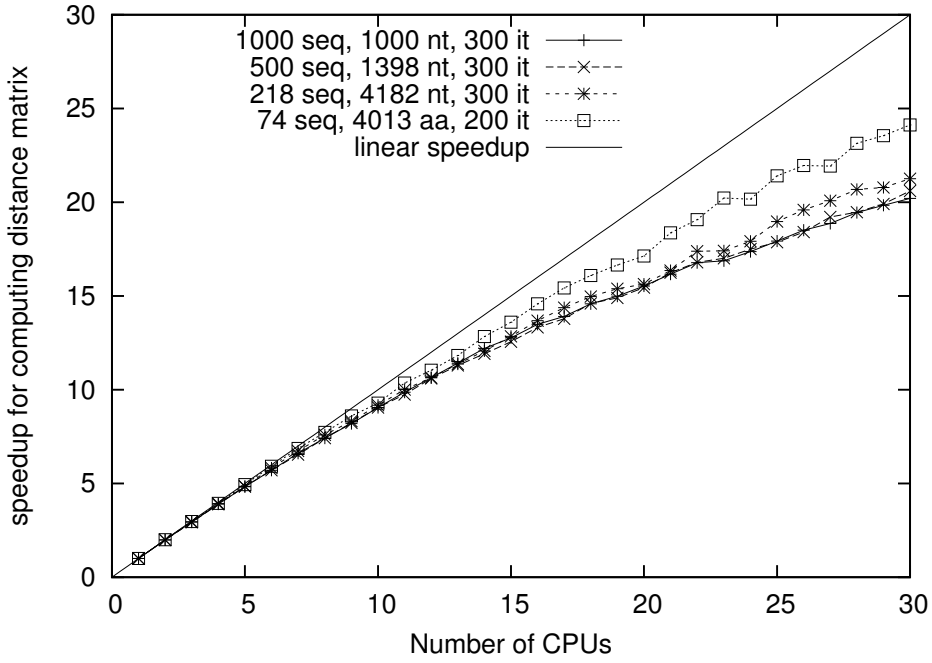

Figure 4.4: Speedup for computing distance matrix.

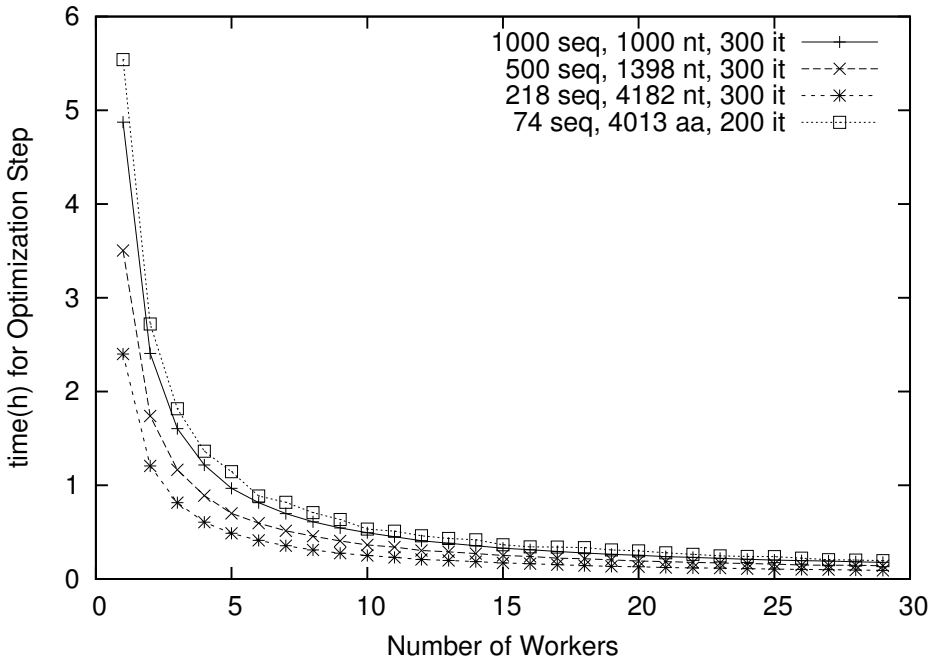

Figure 4.5: Running time for optimization step.

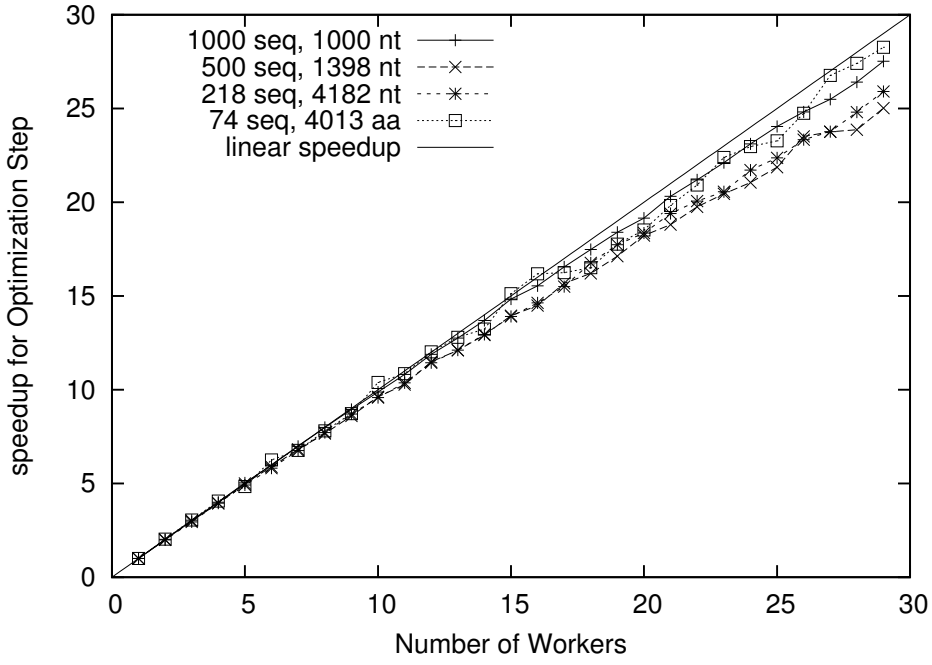

Figure 4.6: Speedup for optimization step.

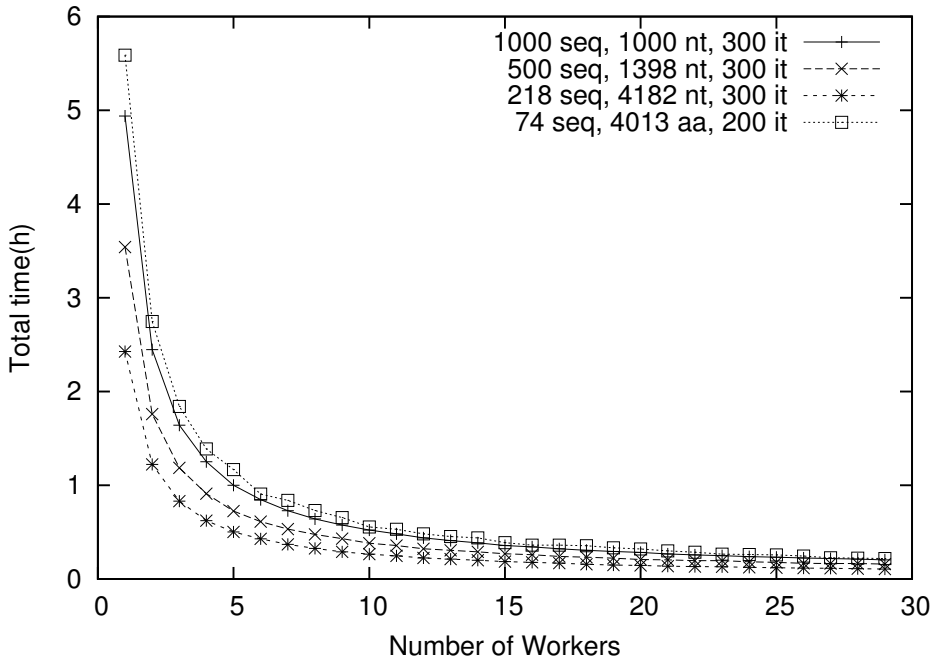

Figure 4.7: Total running time.

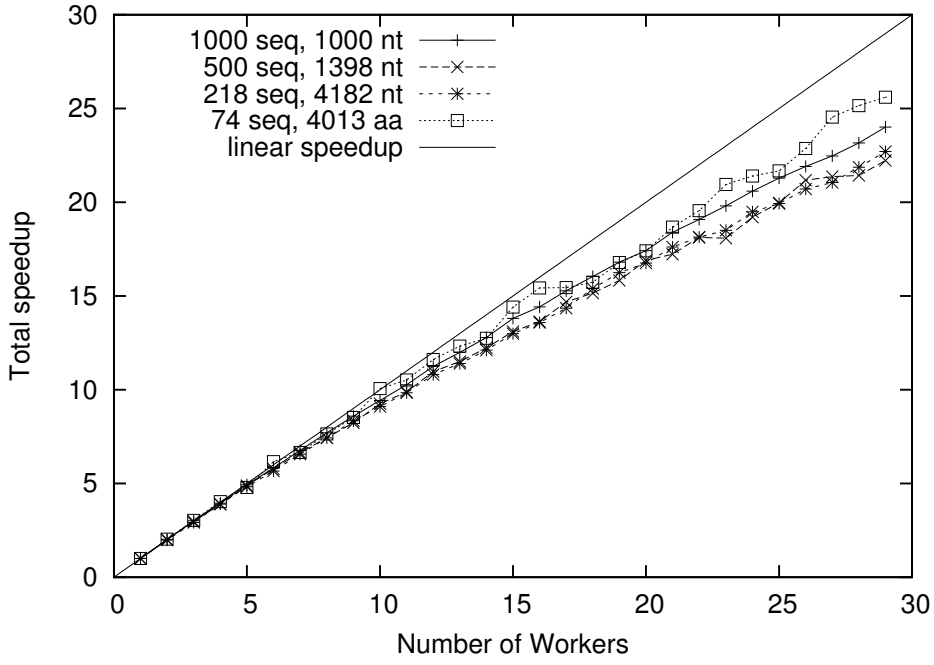

Figure 4.8: Total Speedup.

CPUs of 2.0GHz each, connected via Gigabit Ethernet. The average running time and speedup from 10 independent runs are recorded to avoid random effects.

Figure 4.3 and 4.4 show the runtime and speedup for the parallelization of distance matrix computation as function in the number of CPUs. The speedup is linear up to 18 processors and the slope is close to the theoretical upper bound, in spite of relatively short runtime (from 3 to 25 seconds). Especially, the analysis on the amino acid data performs better than on DNA data. This results from smaller size of the matrix reducing the communication overhead and longer pairwise distance calculation due to higher model complexity and the number of distinct site patterns (Table 3.1).

Figure 4.5 and 4.6 display the running time and speedup for the parallelization of optimization step as function of the number of workers. Figur 4.6 shows that the speedup of the OS is almost linear. The speedup ranges from 25.0 to 28.3 with 29 worker processes. Moreover, a saturation effect is not apparent, as compared with the distance matrix computation.

Even for complete runs of the program (Figure 4.7 and 4.8), including sequential code from the initial step, the speedup drops only slightly, i.e., ranging from 22.2 to 25.6 for 29 workers, again without any recognizable saturation. Despite its small number of sequences, the protein data scales similar to the largest DNA dataset due to longer computation times per iteration (112s compared to 56s) which reduces the communication/computation ratio.

4.4 Accuracy Analysis

As explained Section 4.2, the dependency structure of the sequential algorithm is omitted in the parallel schema: workers try to improve their local T_{worker}, which can be, at some time point, worse than the global T_{master}. The reason is that we do not want to waste workers' effort by simply forcing them to stop ongoing computation everytime one of them discovers a better tree, since it is also possible that the current optimization ends up in a new best tree.

Figure 4.9 and 4.10 illustrate the resulting likelihoods of all iterations during one

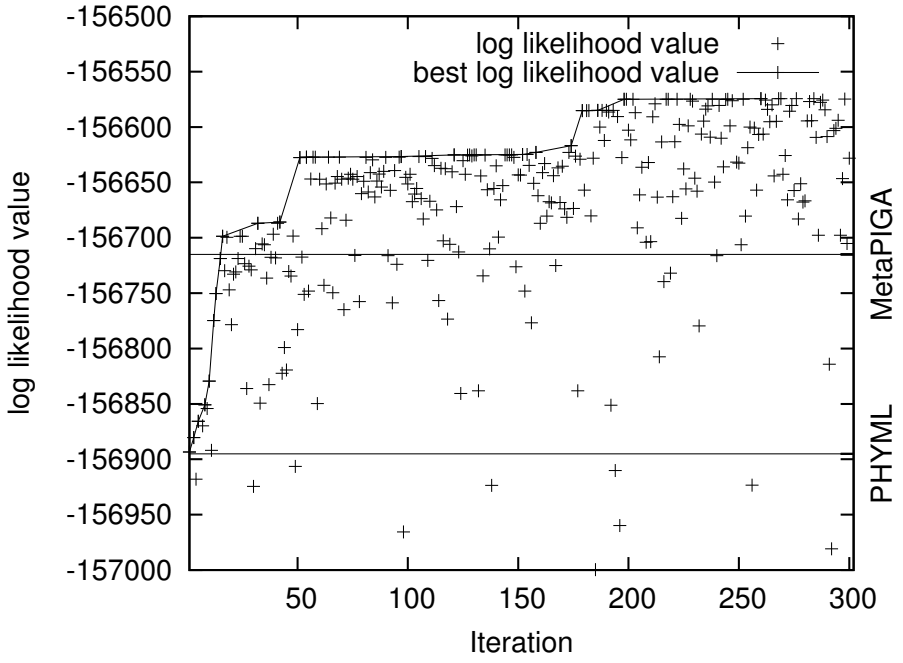

Figure 4.9: Resulting Log-likelihood of a sequential run for dataset *ssu rRNA*.

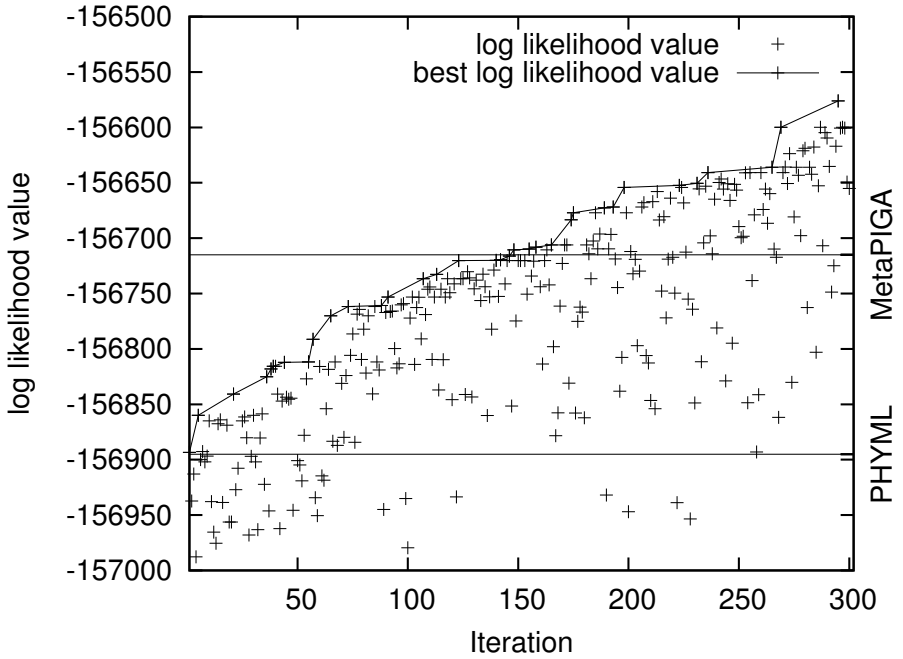

Figure 4.10: Resulting Log-likelihood of a parallel run with 10 workers for dataset *ssu rRNA*.

Table 4.1: Best results obtained running on 58 CPUs.

Properties	rbcL (500 seq)	ssu rRNA (218 seq)
Log-likelihood	-99,977	-156,545
Runtime on 58 CPUs	15h:27m	3h:01m
Runtime on single CPU	36 days	7 days
#Iterations	76,693	21,437
Average runtime 1-CPU/iteration	≈40s	≈30s
#Trees better than PHYML	66,361(86%)	20,579(95%)
#Trees better than MetaPIGA	58,854(76%)	15,632(72%)

sequential run and during one 11-CPUs run respectively. Figure 4.9 indicates several slopes in which the likelihoods go up very fast, but also several plateaus where the best trees are almost unchanged. Figure 4.10, on the contrary, shows the gradual increase in likelihood: Ten workers were working together and finally produced a roughly same result as one. However, they reached the plateau level in about one seventh of the time. Spending a number of additional iterations will lead to the same plateau much earlier than in the sequential run.

4.4.1 Hunting for Best Trees

The efficient parallel implementation enables us to more thoroughly analyze the data by increasing the number of iterations. We re-analyzed the two well known datasets *ssu rRNA* and *rbcL* on a Linux cluster with 58 CPUs of either 1.8GHz or 2.0GHz each. We also turned on the stopping rule (Vinh and von Haeseler, 2004, see also section 2.4.2) to make the program find out automatically when to stop. In the analysis, we compared the results to those of PHYML (Guindon and Gascuel, 2003) and MetaPIGA (Lemmon and Milinkovitch, 2002), two other fast methods to reconstruct large trees, which were also used in the comparison by Vinh and von Haeseler (2004).

Figure 4.11 show the log-likelihood values at every iteration and Table 4.1 summarizes the outcomes. For the ssu rRNA data we achieved a tree with log-likelihood

of $-156,545$, which is 59 units higher than the value reported in Vinh and von Haeseler (2004). The analysis finished after 3 hours, while one would need roughly 7 days if only running the analysis on a single CPU. IQPNNI also found numerous better than PHYML and MetaPIGA.

For the rbcL data, the best tree found so far by pIQPNNI has $-99,977$ as log-likelihood value (34 units higher than the recorded value by Vinh and von Haeseler (2004)). It needed 15.5 hours to ensure a confidence of 95%. That is to imagine, it would take us no less than 36 days to achieve the same result if only using one CPU.

Interestingly, Figure 4.11 indicates a sign of some suboptimal trees, presenting in three recognizable horizontal lines: one at the same level as PHYML did and two others at level of -100,100 and -100,150 approximately. At these levels the method frequently ends up and is only able to escape from these after considerable number of iterations.

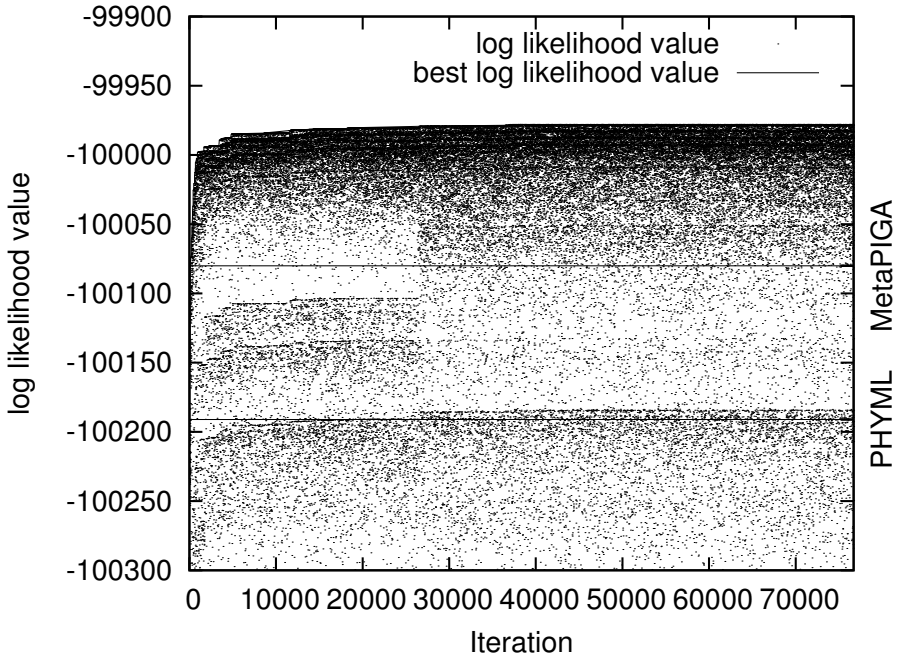

Figure 4.11: Resulting Log-likelihood for dataset *rbcL*: running on 58 CPUs.

Chapter 5

Conclusions and Future Work

The work presented in this thesis has shown that the running time of the IQPNNI algorithm could be substantially decreased by two means:

Sequential improvements (chapter 3): We have examined the original program with profiling technique to locate inefficient codes and applied suitable modifications. It has been further accelerated by employing Newton's method instead of Brent's method to optimize branch lengths of the phylogenies. Moreover, for complex models of substitution, the parameter estimations have been altered by a combination of Newton with BFGS algorithm. These improvements have doubled the speed of the sequential implementation.

Parallelization (pIQPNNI, chapter 4): The improved version was furthermore parallelized using Message Passing Interface. In the initial step, the calculation of the pairwise distance matrix is equally distributed to all processes. For the optimization step (consuming at least 90% of the runtime), we use a master/worker scheme, in which workers perform all the computations and after that send back the results to the master. The master is only in charge of monitoring the traffic, summarizing the global best phylogenies and updating local best trees at workers whenever necessary. We used non blocking communication ensuring that the transmission will not disturb any worker. The introducing of global/local best trees has not kept the dependency structure of the sequential paradigm, in which each iteration only acts on a unique cur-

rent best tree. The parallel performance has therefore been double checked for both accuracy and speedup. Extensive tests showed that the accuracy remained little affected due to the random property of the method. Moreover, we obtained a near linear speedup for both the computation of distance matrix and the optimization step with up to 30 CPUs. This resulted in a near optimal speedup of the whole program without showing any saturation effect. This scaling behavior can be explained by the coarse-grained structure of the algorithm (each OS takes about 30 to 40 seconds, see Table 4.1, which is a lot more than enough for a single data transmission from the worker to the master) and the communication/computation ratio is minimized.

Let us take a look at an example, the analysis time of the 1000-sequences dataset was reduced from 9h:29 to 7h:03 by tuning techniques, then further down to 4h:43 by replacing Brent's for Newton's method. Both enhancements saved us 50% of the runtime. Now putting the job onto a 10-CPUs cluster produced the same result in only 0h:30! This runtime reduction and scaling behavior suggest that pIQPNNI is well-fitted for analyzing large datasets.

Future Work

We believe that this parallelization scheme is suitably and efficiently executed on heterogeneous platforms (see section 4.4.1 where we used a Linux cluster with two kinds of CPUs of 1.8GHz and 2.0GHz). However, a more careful test should be carried out to make sure this property. Our current research also opens some perspectives in which the program can still be enhanced and extended.

First, the sequential program consumes lots of memory. For instance, the analysis on 1000 sequence data needed about 500 MB! This is because of the way it stores all the partial likelihoods in order to evaluate the tree likelihood. This could be improved by applying better mechanism to manage the memory and to release unnecessary information whenever possible.

Second, the parallel implementation can be tested with more than 30 CPUs. It is because of the fact that our testing parallel environment is restricted and not

allowed for a larger configuration. However, this issue could be easily conducted to verify the efficiency and to see whether a saturation effect occurs.

Third, the current parallel version has only been available on MPI. This would be working fine but does not take advantage of the many hybrid supercomputers structure, e.g., cluster of SMPs. Future direction would be to integrate both MPI and OpenMP to create a hybrid program. It will certainly benefit from a large scale parallel execution.

Bibliography

Addario-Berry, L., Chor, B., Hallett, M., Lagergren, J., Panconesi, A. and Wareham, T. (2004) Ancestral maximum likelihood of evolutionary trees is hard. *J. Bioinf. Comput. Biol.*, **2**, 257–271.

Altekar, G., Dwarkadas, S., Huelsenbeck, J. P. and Ronquist, F. (2004) Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference. *Bioinformatics*, **20**, 407–415.

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

Brent, R. P. (1973) *Algorithms for Minimization without Derivatives*. Prentice Hall, Englewood Cliffs, New Jersey, USA.

Bruno, W. J., Socci, N. D. and Halpern, A. L. (2000) Weighted neighbor joining: A likelihood-based approach to distance-based phylogeny reconstruction. *J. Mol. Evol.*, **17**, 189–197.

Chase, M., Soltis, D., Olmstead, R., Morgan, D., Les, D., Mishler, B., Duvall, M., Price, R., Hills, H., Qiu, Y.-L., Kron, K., Rettig, J., Conti, E., Palmer, J., Manhart, J., Sytsma, K., Michaels, H., Kress, W. J., Karol, K., Clark, W. D., Hedren, M., Gaut, B., Jansen, R., Kim, K.-J., Wimpee, C., Smith, J., Furnier, G., Strauss, S., Xiang, Q.-Y., Plunkett, G., Soltis, P., Swensen, S., Williams, S., Gadek, P., Quinn, C., Eguiarte, L., Golenberg, E., Learn, G., Graham, S., Barrett, S., Dayanandan, S. and Albert, V. (1993) Phylogenetics of seed plants: an analysis of nucleotide sequences from the plastid gene rbcL. *Annals of the Missouri Botanical Garden*, **80**, 528–580.

Chor, B. and Tuller, T. (2005) Maximum likelihood of evolutionary trees is hard. In *Proceedings of the 9th Annual International Conference on Research in Computational Molecular Biology (RECOMB 2005)*, volume 3500 of *Lecture Notes in Computer Science*, pp. 296–310, ACM Press, New York, USA.

Dagum, L. and Menon, R. (1998) OpenMP: An industry-standard API for shared-memory programming. *IEEE Comput. Sci. Eng.*, **5**, 46–55.

Darwin, C. (1859) *On the Origin of Species by Means of Natural Selection*. John Murray, London.

Day, W. H. E. (1987) Computational complexity of inferring phylogenies from dissimilarity matrices. *Bull. Math. Biol.*, **49**, 461–467.

Day, W. H. E. and Sankoff, D. (1986) Computational complexity of inferring phylogenies by compatibility. *Syst. Zool.*, **35**, 224–229.

Edgar, R. C. (2004) MUSCLE: multiple sequence alignment with high accuracy and high throughput. *Nucl. Acids Res.*, **32**, 1792–1797.

Edwards, A. W. F. and Cavalli-Sforza, L. L. (1964) Reconstruction of evolutionary trees. In Heywood, V. H. and McNeill, J. (eds.), *Phenetic and Phylogenetic Classification*, pp. 67–76, Systematics Association, London, UK.

Felsenstein, J. (1978) The number of evolutionary trees. *Syst. Zool.*, **27**, 27–33.

Felsenstein, J. (1981) Evolutionary trees from DNA sequences: a maximum likelihood approach. *J. Mol. Evol.*, **17**, 368–376.

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

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

Gascuel, O. (1997) BIONJ: An improved version of the NJ algorithm based on a simple model of sequence data. *Mol. Biol. Evol.*, **14**, 685–695.

Gill, P. E., Murray, W. and Wright, M. H. (1981) *Practical Optimization*. Academic Press, London, UK.

Graham, R. L. and Foulds, L. R. (1982) Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Math. Biosci.*, **60**, 133–142.

Gropp, W., Huss-Lederman, S., Lumsdaine, A., Lusk, E., Nitzberg, B., Saphir, W. and Snir, M. (1998) *MPI: The Complete Reference - The MPI Extensions*, volume 2. 2nd edition, The MIT Press, Cambridge, Massachusetts.

Guindon, S. and Gascuel, O. (2003) A simple, fast, and accurate algorithm to estimate large phylogenies by maximum likelihood. *Syst. Biol.*, **52**, 696–704.

Hagerup, T. (1997) Allocating independent tasks to parallel processors: An experimental study. *J. Parallel Distrib. Comput.*, **47**, 185–197.

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

Hillis, D. M. and Wiens, J. J. (2000) Molecules versus morphology in systematics: Conflicts, artifacts and misconceptions. In Wiens, J. J. (ed.), *Phylogenetic analysis of morphological data*, pp. 1–19, Smithsonian Institution press, Washington and London.

Hwang, K. and Xu, Z. (1998) *Scalable parallel computing: technology, architecture, programming*. McGraw-Hill Series in Computer Engineering, New York, USA.

Kanz, C., Aldebert, P., Althorpe, N., Baker, W., Baldwin, A., Bates, K., Browne, P., van den Broek, A., Castro, M., Cochrane, G., Duggan, K., Eberhardt, R., Faruque, N., Gamble, J., Garcia Diez, F., Harte, N., Kulikova, T., Lin, Q., Lombard, V., Lopez, R., Mancuso, R., McHale, M., Nardone, F., Silventoinen, V., Sobhany, S., Stoehr, P., Tuli, M. A., Tzouvara, K., Vaughan, R., Wu, D., Zhu, W., and Apweiler, R. (2005) The EMBL nucleotide sequence database. *Nucl. Acids Res.*, **33**, D29–D33.

Keane, T. M., Naughton, T. J., Travers, S. A. A., McInerney, J. O. and McCormack, G. P. (2005) DPRml: distributed phylogeny reconstruction by maximum likelihood. *Bioinformatics*, **21**, 969–974.

Lemmon, A. R. and Milinkovitch, M. C. (2002) The metapopulation genetic algorithm: An efficient solution for the problem of large phylogeny estimation. *Proc. Natl. Acad. Sci. USA*, **99**, 10516–10521.

Morgenstern, B. (1999) DIALIGN 2: improvement of the segment-to-segment approach to multiple sequence alignment. *Bioinformatics*, **15**, 211–218.

Notredame, C., Higgins, D. and Heringa, J. (2000) T-COFFEE: A novel method for multiple sequence alignments. *Journal of Molecular Biology*, **302**, 205–217.

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

Press, W. H., Flannery, B. P., Teukolsky, S. A. and Vetterling, W. T. (1992) *Numerical Recipes in C: The Art of Scientific Computing*. 2nd edition, Cambridge University Press, Cambridge.

Saitou, N. and Nei, M. (1987) The neighbor–joining method: A new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, **4**, 406–425.

Schmidt, H. A., Strimmer, K., Vingron, M. and von Haeseler, A. (2002) TREE-PUZZLE: Maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, **18**, 502–504.

Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W. and Dongarra, J. (1998) *MPI: The Complete Reference - The MPI Core*, volume 1. 2nd edition, The MIT Press, Cambridge, Massachusetts.

Spencer, M., Susko, E. and Roger, A. J. (2005) Likelihood, parsimony, and heterogeneous evolution. *Mol. Biol. Evol.*, **22**, 1161–1164.

Stamatakis, A. and Ludwig, T. (2004) The AxML program family for phylogenetic tree inference. *Concurr. Comput.-Pract. Exp.*, **16**, 975–988.

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

Strimmer, K. and von Haeseler, A. (2003) Nucleotide substitution models. In Salemi, M. and Vandamme, A.-M. (eds.), *The Phylogentic Handbook*, pp. 72–87, Cambridge University Press, Cambridge, UK.

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.

Tateno, Y., Takezaki, N. and Nei, M. (1994) Relative efficiencies of the maximum–likelihood, neighbor–joining, and maximum–parsimony methods when substitution rate varies with site. *Mol. Biol. Evol.*, **11**, 261–277.

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

Thompson, J. D., Higgins, D. G. and Gibson, T. J. (1994) CLUSTAL W: Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, positions–specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, **22**, 4673–4680.

Trelles, O. (2001) On the parallelisation of bioinformatics applications. *Brief. Bioinform.*, **2**, 181–194.

Vandamme, A.-M. (2003) Basic concepts of molecular evolution. In Salemi, M. and Vandamme, A.-M. (eds.), *The Phylogentic Handbook*, pp. 1–23, Cambridge University Press, Cambridge, UK.

Vinh, L. S. and von Haeseler, A. (2004) IQPNNI: Moving fast through tree space and stopping in time. *Mol. Biol. Evol.*, **21**, 1565–1571.

Vinh, L. S. and von Haeseler, A. (2005) Shortest triplet clustering: reconstructing large phylogenies using representative sets. *BMC Bioinformatics*, **6**.

Waterman, M. S. (1995) *Introduction to Computational Biology*. Chapman and Hall, London.

Whelan, S. and Goldman, N. (2001) A general empirical model of protein evolution derived from multiple protein families using a maximum likelihood approach. *Mol. Biol. Evol.*, **18**, 691–699.

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