NTNU
Norwegian University of
Science and Technology

Inferring Phylogenies Using Evolutionary Algorithms

A maximum likelihood approach for constructing phylogenetic trees from molecular data

Erlend Heggheim Hamberg

Master of Science in Computer Science
Submission date: June 2011
Supervisor: Keith Downing, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

The thesis aims to evaluate the use of evolutionary algorithms (EAs) for inferring phylogenies from molecular data. These methods will be compared with conventional software packages and with earlier work on using EAs for phylogeny inference – both with respect to the likelihood of the trees found and the computational costs.

Assignment given:	17 January, 2011
Supervisor:	Keith Downing

Abstract

This thesis has evaluated the use of the computationally expensive maximum-likelihood (ML) method coupled with an evolutionary algorithm (EA) for the problem of inferring evolutionary relationships among species (phylogenies) from molecular data. ML methods allow using all the information from molecular data, such as DNA sequences, and have several beneficial properties compared to other methods. Evolutionary algorithms is a class of optimization algorithms that often perform well in complex fitness landscapes. EAs are also proclaimed to be easy to parallelize, an aspect that is increasingly more important.

A parallel EA system has been implemented and tested on a cluster for the task of phylogeny inference. The system shows promising results and is able to utilize processors of a massively parallel system in a transparent manner.

Acknowledgements

I am grateful for having been allowed to combine two of my academic interests in this project: evolutionary genetics and artificial intelligence methods. The assignment was originally suggested by Helge Langseth, but he was away for a sabbatical and Keith Downing was kind enough to supervise this project – perhaps after seeing how much I wanted to work with this. Jenny Hagenblad from the Department of Biology at NTNU helped me with the biology theory and has been a de facto second supervisor during my work with this thesis.

Contents

1 Introduction **1**

2 Background **5**

2.1 Phylogenetics 5

2.1.1 Genetic Code 5

2.1.2 Alignment 7

2.1.3 Phylogenetic Trees 8

2.1.4 Phylogenetic Inference 10

2.1.5 Probabilistic Models of Evolution 12

2.2 Evolutionary Algorithms 14

2.2.1 Artificial Evolution 15

2.2.2 Genetic Algorithms 17

2.2.3 Representing Trees 19

2.3 Maximum Likelihood 21

2.4 File Formats 22

2.4.1 Phylogenetic Trees 22

2.4.2 Sequence Data 23

2.5 Parallel Execution 25

2.6 Related Work 25

2.6.1 GAML (1998) 25

2.6.2 Cotta & Moscato (2002) 27

3 Methods **29**

3.1 Calculating the Likelihood of a Tree 29

3.1.1 A Recursive Algorithm for Computing $L_k^{(i)}(s)$ 31

3.1.2 Evaluation of Computational Cost 33

3.2 Unrootedness 34

3.3 Sources of Error 34

3.4 GA Operators for Phylogenetic Trees 35

4 Implementation **37**
4.1 Overview . . . 37
4.2 Implementation details . . . 38
4.2.1 EA System . . . 38
4.2.2 Tree Evaluator . . . 38
4.2.3 Data Structures . . . 38
4.2.4 Differential Reproduction . . . 41
4.2.5 GA Operators for Phylogenetic Trees . . . 41
4.2.6 Reading Sequence Data . . . 41
4.2.7 Displaying Trees . . . 42
4.2.8 Testing . . . 42
4.3 Parallelization . . . 42
4.4 Prune-Delete-Graft with Branch Lengths . . . 43
4.5 Tree Representation . . . 43
4.6 Using PHYML to Evaluate Likelihoods . . . 45

5 Results **46**
5.1 Twenty Mammalian Mitochondrial Genomes . . . 48
5.1.1 Runs . . . 49
5.1.2 Dissecting a Run . . . 49
5.1.3 Differential Reproduction . . . 52
5.2 Ten Ray-Finned Fishes, Ten Mammals . . . 55
5.3 Running Time . . . 58

6 Conclusion and Further Work **60**
6.1 Further Work . . . 62

Bibliography **65**

A Data sets **69**
A.1 `mamm20` . . . 69
A.2 euteleostomi20 . . . 71

B PHYML Inferred Trees **73**

C Fitness Values **76**

D Distributed computations on a cluster **80**
D.1 Interactively running a job . . . 80
D.2 Queueing a job . . . 81

E Preparing the Sequence Data **83**

List of Figures

1.1 The three possible rooted, bifurcating trees for a phylogeny of three species . . . 1
1.2 Overview of the process of phylogenetic inference using an EA 3

2.1 Double stranded DNA . . . 6
2.2 Illustration of an insertion mutation . . . 7
2.3 Aligned DNA sequences . . . 7
2.4 An unrooted and a rooted tree . . . 9
2.5 Sample tree showing changes . . . 11
2.6 Expected number of changes along branches . . . 12
2.7 Transitions vs transversions . . . 13
2.8 Overview of an evolutionary algorithm . . . 16
2.9 Recombination example . . . 18
2.10 Prefix notation . . . 19
2.11 Prüfer sequence coding . . . 20
2.12 Example tree illustrating the Newick format . . . 23
2.13 An example FASTA file with two sequences. . . . 24
2.14 An example interleaved PHYLIP file with two sequences. . . . 24
2.15 Distributed fitness evaluation . . . 26

3.1 Hypothetical phylogenetic tree . . . 30
3.2 One subtree of the hypothetical phylogenetic tree . . . 32
3.3 Recombination of trees using *Prune-Graft-Delete* . . . 36

4.1 Prune-Graft-Delete on a tree with branch lengths . . . 44
4.2 Example genome . . . 45

5.1 The best tree found for the `mamm20` data set . . . 47
5.2 Fitness plots for the `mamm20` data set. . . . 50
5.3 Fitness plot for one of the runs on the `mamm20` data set. . . . 51
5.4 Number of tree topologies. . . . 52
5.5 "Success rates" for mutations and recombinations . . . 53

5.6 Fitness plots for the mamm20 data set with differential reproduction. 54
5.7 Number of tree topologies when using differential reproduction. 55
5.8 The best tree found for the mamm20 data set with differential reproduction, with a log likelihood of−152018. 56
5.9 The best tree found for the ray10mamm10 data set with differential reproduction, with a log likelihood of−158556. 57
5.10 Fitness plots for the ray10mamm10 data set. 59

6.1 The pulley principle 63

B.1 The phylogenetic tree inferred by PHYML for the mamm20 data set. 74
B.2 The phylogenetic tree inferred by PHYML for the ray10mamm10 data set. 75

D.1 PBS jobscript example 82

Chapter 1

Introduction

Phylogenetics is the science of determining the evolutionary relationships among organisms. Knowledge about these relationships is central to our understanding of how life has developed. The evolutionary history of a species or a set of species is called a *phylogeny* and is usually visualized as a *phylogenetic tree*. These trees represent hypotheses for the evolutionary history and can help us understand the history of mutations and aid in the understanding of biological processes [14]. Phylogenetic trees are classified based on whether they are rooted or not and whether they are bifurcating (each ancestor spawns two lineages) or multi-furcating (an ancestor can spawn more than two lineages). Fig. 1.1 shows the three possible rooted, bifurcating trees describing the evolutionary relationship among three species. Phylogenetic trees are used to trace the evolution on the level of taxa, species or individual genes.

Phylogenetic inference is the problem of reconstructing a phylogeny from a set of data – usually molecular data such as DNA sequences. The problem of reconstructing the optimal phylogeny from a data set is an NP-complete problem [23]. Exact techniques for inferring the most likely tree exist, but

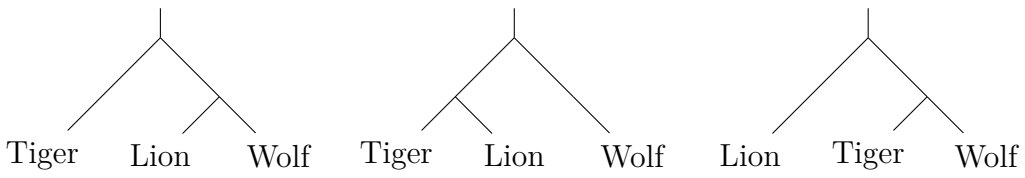


Fig. 1.1: There are three possible rooted, bifurcating trees for a phylogeny of three species. This follows intuitively from the fact that two of the species necessarily must be more closely related to each other than to the third species.

are computationally infeasible even for trees of 30–40 species [6] due to the enormous number of possible trees. Heuristic search algorithms are therefore often applied to find an approximation to the most likely phylogenetic tree [6, 23, 25, 27].

Evolutionary algorithms (EAs) is a class of heuristic optimization algorithms. These algorithms are computational systems inspired by biological evolution in which candidate solutions are artificially "evolved" and features from the best solutions have higher probability of "surviving" from one generation to the next. Earlier research has produced some encouraging – albeit preliminary – results [23, 6].

Due to the way computations in an EA are localized and independent of other parts of the system, EAs lend themselves easily to parallel computing. This aspect has become increasingly more important as power consumption and heat generation from processors has led to parallel computing becoming the dominant computation paradigm in recent years [2]. Even consumer computers today have multiple processor cores, and it is common to use so-called *clusters* of interlinked computers to perform resource-demanding calculations.

This thesis aims to evaluate the use of EAs for inferring phylogenies from molecular data. To evaluate the likelihood of a tree a maximum likelihood (ML) approach is used. Using ML to evaluate a tree's likelihood is computationally very expensive compared to other methods, but has several beneficial statistical properties that become even more important as the amount of data grows. As the computational barriers to maximum likelihood estimation has been removed by faster computers, most of the focus of statistical methods for phylogeny inference has been shifted to concentrate on likelihood methods [10].

A high-level overview of the process of phylogeny inference with the EA system is shown in Fig. 1.2 on the next page. DNA sequences from the species we want to infer a phylogeny for is first *aligned*. Alignment is a process where insertions and deletions – called *frame shift* mutations – in sequences are accounted for so that homologous parts of the sequences "line up". These aligned sequences are then used as input to the EA system which outputs one or more trees that are hypotheses for the true tree. To find good trees, the EA uses an iterative process where different trees are generated and evaluated ("EA cycle"). A parallel EA system has been implemented as part of this thesis.

> *The goal of this thesis is to evaluate the use of EAs to infer phylogenies. This will be viewed in light of the computational resources needed and the results produced. Both the results and*

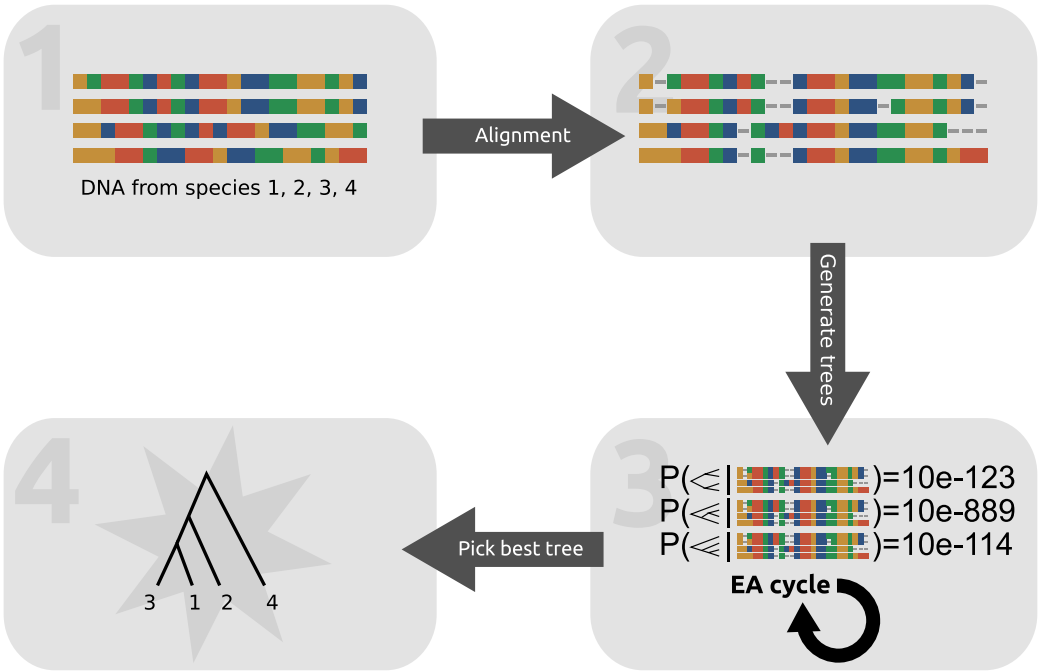


Fig. 1.2: Overview of the process of phylogenetic inference using an EA. In the first step, the DNA sequences are aligned. These aligned sequences are then used as the data in the leaf nodes of a set of randomly generated trees. The likelihood of these trees given the data are then calculated and these likelihoods are used as the fitness values for trees in the iterative "EA cycle". After a given number of generations, the best tree found is then taken to be an approximation to the most likely phylogenetic tree for the species being looked at.

the computational resources used will be compared to conventional software used in the field today. The issue of parallel computing will be given extra weight and the EA system should be able to utilize a parallel computer system.

This thesis is structured in the following manner:

- This introduction is meant to serve as a $10,000$-metre view of the problem area and hopefully make it easier for the reader to see the whole picture as each part is more thoroughly explained in the following chapters.
- The background chapter presents the background information on the concepts and the biology and genetics knowledge needed to understand the research area.
- The third chapter gives a detailed explanation of the calculation of likelihood for a phylogenetic tree given molecular data. This likelihood value is used as a tree's *fitness* in the EA system.
- Chapter four and five then show the implementation and the results of using this to infer trees for real-life data sets.
- The last chapter summarizes the results of testing the system and lists further work that should be done within this area. The merits of using EA for phylogenetic inference is discussed.

Chapter 2

Background

This chapter aims to provide the reader with the background knowledge needed to understand how phylogenetic inference is done. A brief introduction to the biology of phylogenetics is given and evolutionary algorithms are explained. A short explanation of the maximum-likelihood method used is also included, along with a brief overview of file formats used in this field. The chapter ends with a look at parallel software and related work done in this field.

2.1 Phylogenetics

A *phylogeny* is a reconstructed evolutionary history for a collection of organisms and is usually presented in the form of a tree. The study of evolutionary relatedness among a set of entities is called *phylogenetics*. In biology this is usually among different species or between groups of related species, but phylogenetic analysis is also used in other areas such as natural language studies in which case the entities looked at will be human languages [4].

In *molecular* phylogenetics, molecular data – such as DNA or RNA sequences – is used to infer the evolutionary relationship between species.

2.1.1 Genetic Code

The genomes of all living organisms except for most viruses are coded with *DNA*. DNA is made up of long chains of nucleotides – molecules consisting of a nitrogenous base, a five-carbon sugar, and one to three phosphate groups. Each nucleotide has one of four possible bases and it is the sequence of these bases in the nucleotides along the DNA molecule that encodes the genetic information. Each position along the DNA strands that holds a nucleotide

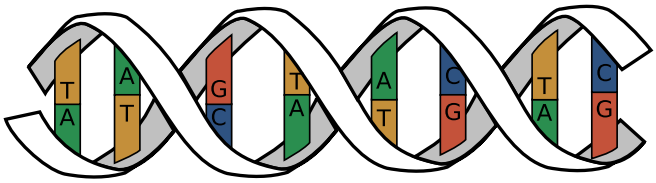


Fig. 2.1: Double stranded DNA. A bonds with T and C bonds with G. Here eight sites are shown.

is referred to as a *site*.

The four bases found in the nucleotides in DNA are *adenine* (A), *cytosine* (C), *guanine* (G) and *thymine* (T). These will simply be referred to by their initial letters when there is no risk of confusion. These bases are divided into two groups: Adenine and guanine are compounds called *purines*, while cytosine and thymine are *pyrimidines*. The genetic code is thus a quaternary (4-ary) code written in the "alphabet" $\langle A, C, G, T \rangle$. This is also true for viruses with *RNA genomes* except that in their genomes uracil (U) occur instead of thymine.

DNA molecules consist of two chains joined by hydrogen bonds between pairs of bases. A bonds with T and C bonds with G in a double helix, as illustrated in Fig. 2.1. Since bases form pairs in this way, the two strands are *complementary* – it is enough to know the sequence of one strand. We refer to connected nucleotides as *base pairs*.

The DNA in cells are organized in physical structures called *chromosomes* in the cell nucleus. The full DNA sequence of an organism (all DNA in all its chromosomes) is called its *genome*. All animal cells (in fact all aerobic eukaryotic cells) also contain a *mitochondrial*[1] *genome*. This is a much smaller DNA genome that is inherited only through the maternal line [33].

As an example, the human genome is around 3 billion DNA base pairs [32], while the human, mitochondrial genome consists of around 16,000 base pairs [33].

"Reading" the DNA from a cell is called *sequencing* and the resulting data is commonly referred to as *sequence data*. There has been rapid advances in sequencing technology the last decades which has led to an explosion in the amount of available sequence data [28].

[1] Plant cells have a *chloroplast* genome.

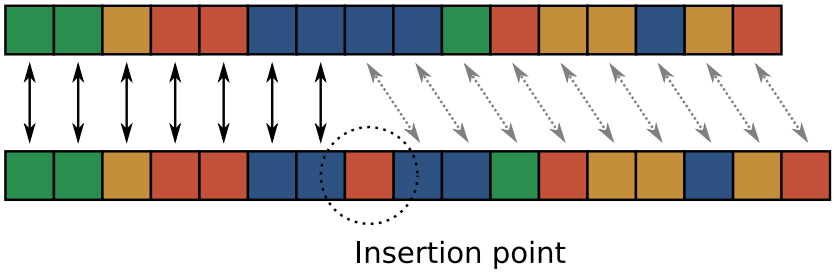


Fig. 2.2: An insertion of an extra nucleotide leads to the loss of the one-to-one correspondence between the sites of the two sequences. A deletion would have the same consequence.

```
CTATCGC---TCATTGATCCAAAAATTT--GATCAAC
ACATCAC---TTATTGATCCAATAATTTTTGATCAAC
CTACCACATTTAATTGATCCAATGACTT--GACCAAC
CTACCACATTTAATTGATCCAATGACTT--GACCAAC
```

Fig. 2.3: Aligned DNA sequences with gaps at sites 8–10 and 29–30.

2.1.2 Alignment

Genetic variation is created by several processes. The ultimate source of variation is *mutation*: changes to the genetic material that are heritable. Another important process is *recombination* which happens when strings of DNA are broken off and joined to another part of the genome. This happens during the formation of sex cells in animals and plants[2] [16]. An insertion or deletion of a base in one of two identical sequences will shift all the following bases one site to the right (insertion) or left (deletion). This small change will thus destroy the $1:1$ agreement between the bases at corresponding sites. This concept is illustrated in Fig. 2.2 If one were to directly perform phylogenetic inference on these sequences they would appear to have diverged much more than is reality.

These problems are remedied by inserting *gaps* in the sequences in order to find the most likely alignment of related sequences. This process is called *sequence alignment* and aims to find the most likely alignment among two or more sequences. Fig. 2.3 shows four sequences that have been aligned by inserting gaps (shown as dashes). Sequence alignment is usually done by software that use probabilistic methods to find the most likely alignment.

One widely used software package for alignment of multiple sequences is

[2]It also happens in other organisms. Recombination also sometimes happen during *mitosis* – when a cell divides itself in half to make two identical copies.

Clustal which is available both as a standalone program and as a free web service at the European Bioinformatics Institute [5].

2.1.3 Phylogenetic Trees

A *phylogenetic tree* is a tree showing the evolutionary relationships between different species or other entities based on similarities with respect to some characteristics. In molecular phylogenetics these characteristics are often DNA or RNA sequences.

Phylogenies are usually presented in the form of a tree because vertical evolution (parents passing on their genetic material to their offspring) is the primary mechanism of inheritance [29]. It is not the only mechanism, however: there is also horizontal transfer of genetic material by means such as horizontal gene transfer between bacteria and some other taxa. A phylogenetic tree is thus only a pragmatic approximation to the real history [29].

Two closely related species – i.e. species that recently split off from their most recent common ancestor (MRCA) – will have accumulated fewer changes between them and their MRCA, and their genomes will have relatively few differences. As the time between them and their MRCA increases, these differences will increase as the two lineages are evolving independently. This principle is used to identify the tree that has the highest likelihood. A good phylogenetic tree needs to account for the amount of changes between the species in the leaf nodes and their ancestors. Two DNA sequences that are very similar will probably share a common ancestor that is relatively close (in time).

It is easy to realize that all living species will be *leaf nodes* in any phylogenetic tree – they cannot be any species' ancestors since they still exist. We only ever observe the DNA of these leaf nodes of the "tree of life", since DNA degrades quickly in nature. Internal nodes in the tree are therefore *hypothetical* ancestors of our "leaf node species". Other lines of evidence (e.g. fossils), when they exist, can be used to evaluate our hypotheses.

The theory of universal common ancestry posits that all living organisms on Earth share a common ancestor. This theory is widely accepted [37] and would mean that *all* organisms are part of one gigantic tree of life.

Rooted vs Unrooted Trees

A phylogenetic tree can be *rooted* or *unrooted*. In a rooted tree there is a unique node corresponding to the most recent common ancestor of all the species in the leaf nodes. In an unrooted tree no assumption is made about ancestry and only the relatedness of the leaf nodes is found. Fig. 2.4 shows

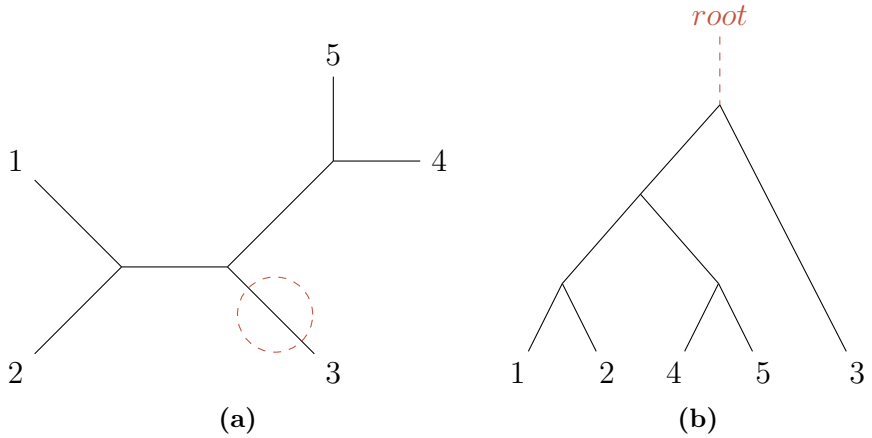


Fig. 2.4: An unrooted (a) and a rooted tree (b). The rooted tree is created from (a) by placing the root along the branch connecting node 3 to the tree (highlighted in red).

an example of an unrooted and one of the possible rooted trees that is found by placing the root along a branch in the unrooted tree.

Most methods for inferring phylogenies infer unrooted trees. Two commonly used methods of rooting an unrooted tree are [10, 19]:

1. By using an *out-group* – a group that is known to be quite closely related to the species under study, but not as closely as the other species are to each other. This helps since we then know that the out-group branched from the root before the other species branched from each other.

2. By using the *molecular clock* hypothesis. This hypothesis posits that the average rate of evolution can be uniform throughout long periods of evolutionary time [16]. This means that the root should be chosen to make the amount of change down to all leaf nodes relatively uniform.

How Many Trees are There?

The number of possible bifurcating trees grows quickly as the number of species considered increases. For n species there are $(2n-3)!/\left(2^{n-2}\,(n-2)!\right)$ possible rooted tree topologies [16]. For 10 species the number of possible rooted trees is $34,459,425$, and for 20 species it is 8.2×10^{21}.

2.1.4 Phylogenetic Inference

Phylogenetic inference is the problem of finding the tree that best represents the evolutionary history of a set of species, given data about the species. *Molecular* phylogenetics has become increasingly feasible as advances in sequencing techniques has led to an accelerating growth of available genetic data.

Exact techniques for doing phylogenetic inference exist, but are computationally infeasible even for trees of 30–40 organisms [6]. This is due to the enormous number of possible trees.

There are numerous methods for constructing phylogenetic trees from molecular data. They can be classified into parsimony methods, distance methods, and likelihood methods [10]. *Parsimony methods* assume that base changes are unlikely events and tries to find the trees that minimize changes. *Distance methods* pre-process the data and make a distance matrix with pairwise distances between sequences and then only use these distances to come up with a tree. Finally, *likelihood methods* try to make use of the full sequence data by formulating a probabilistic model of evolution and using statistical methods such as maximum likelihood estimation.

Parsimony Methods

Parsimony methods seek to find the tree that minimize the required changes between states (bases) per site – with the underlying assumption that the tree with the fewest changes is the most likely. This method thus assumes that such mutations are unlikely events. Given a tree we can count the changes by looking at one site at the time and assign each possible base (A, C, G, and T) to the root node and see how many changes are required down the tree given the observed data. In Fig. 2.5 on the next page this is done for a simple tree of five species. If we assume that the root node has state C, the 3 changes shown are required to explain the observed data. By doing the same for A, G, and T, too, it is found that 3 is the minimum amount of state changes.[3]

Parsimony methods are computationally efficient and work very well if the amount of change over the time period considered is small. However, if different lineages have sufficiently unequal amounts of change, it has been shown that these methods can be inconsistent, converging to a wrong tree as more sequences are considered [8]. One common problem with parsimony methods is *long branch attraction*. This happens when two lineages end up with the same state change for a site. The probability of this increases as

[3]Assuming A, G, or T for the root gives 4 changes in all three cases.

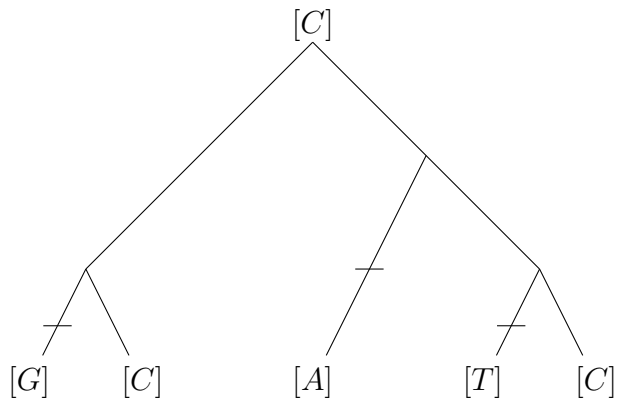


Fig. 2.5: A tree for five species showing their state at one site. The minimum amount of changes needed to explain this tree is three. The branches marked with a horizontal bar show where these changes could have taken place when we have observed the state C at the root.

branches get longer. Parsimony will erroneously consider these two changes as one change in their most recent common ancestor [10].

Distance Methods

Distance methods start by reducing the data to a distance matrix, a two-dimensional matrix of distances between each pair of species. The values M_{ij} thus represents the evolutionary distance between i and j [6]. These distances can be thought of as the branch length separating each pair of species. Only the data in this distance matrix is then used to infer a tree – with methods such as least square minimization. This excludes all information from higher-order combination of states. This principle seems as if it would not work very well, but computer simulations have shown that the amount of information that is lost by working only with pairwise distances is small [10]. It has been shown, however, that clustering methods based on pairwise distances can sometimes give inconsistent estimates if the rates of evolution are unequal in different lineages [8].

Maximum Likelihood

A maximum likelihood (ML) approach to phylogenetic inference tries to use all of the available sequence data. This is done by constructing a probabilistic model of evolution and then use the statistical method of maximum likelihood to assign a likelihood to a tree. This method requires more computation than

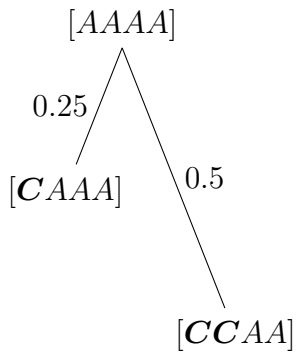


Fig. 2.6: Along a branch of length 0.25 for four bases, the expected number of changes is one. Along a branch of length 0.5, two of the four bases are expected to have changed. (Changes in bases compared to the ancestor are in bold letters.)

parsimony and distance methods, but have several desirable properties. ML maximization will converge to the correct parameter values and the smallest possible variance around these values as the amount of data grows [10]. A brief explanation of Maximum Likelihood is given in Section 2.3 on page 21.

2.1.5 Probabilistic Models of Evolution

In order to use a maximum likelihood approach, we need a model of DNA evolution. Such a model is a statistical description of the process of substitution in nucleotide sequences and can tell us the expected frequency of the different bases at equilibrium and the probability of a specific base change over a branch of length t:

$$P\left(j \mid i, t\right), \tag{2.1}$$

i.e. the probability of seeing base j at the end of a branch of length t given that the start of the branch has base i.

Complex models do a better job of approximating the biological processes but at the expense of more parameters that must be estimated and higher computational requirements [29].

Instead of measuring branch lengths in (millions of) years, they are defined to be the *expected nucleotide substitutions per site*. This means that if we have a branch of length 0.5 between an ancestor and a descendant, we expect half the sites to have undergone changes. An idealized example where the expected number of changes is equal to the actual number of changes is shown in Fig. 2.6 .

There are several models of nucleotide evolution in use, The Jukes-Cantor model is very simple but still widely used, while the Kimura model takes into

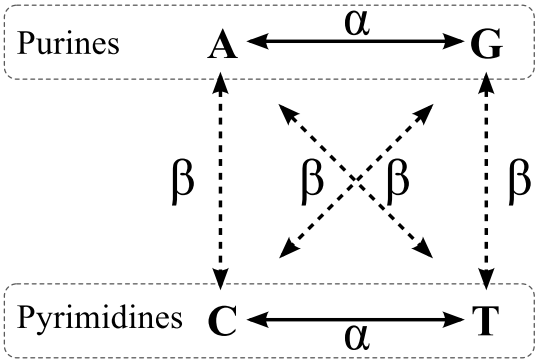


Fig. 2.7: *Transitions* (solid arrows) vs *transversions* (dotted arrows). The probabilities of a state change in Kimura's two-parameter model are α for transitions and β for transversions.

account that not all changes are equally probable.

Jukes-Cantor

The *Jukes-Cantor* model uses a uniform probability of a base changing into any other base. The likelihood of changing from one base to another is simply $u/3$ where u is the overall rate of change [10]. This model is in most cases too simple. There is usually a significant difference between *transitions*, which are changes from one purine or pyrimidine to another of the same group (e.g. $A \leftrightarrow G$), and *transversions,* which are changes from a purine to a pyrimidine or vice versa (e.g. $A \leftrightarrow C$). The ratio of transitions to transversions can be as high as 10 for some mitochondrial DNAs [10].

Kimura's Two-Parameter Model

Kimura's two-parameter model (often called *Kimura80* or simply *K80* after the year it was published) is a slightly more advanced model that takes into account that the rates of change may vary between transitions and transversions. The rate of transitions is denoted α and the rate of transversions is denoted β. This model is illustrated in Fig. 2.7 . (Note that the Jukes-Cantor model is simply a special case of Kimura's model where $\alpha = \beta$.) From any nucleotide there is one change that causes a transition (e.g. $A \rightarrow G$) and two changes that cause a transversion (e.g. $A \rightarrow C$, $A \rightarrow T$). The ratio of transitions to transversions, denoted R, is thus $R = \alpha/2\beta$.

Since we have two possible transversions, each with probability β, and one possible transition with probability α from any base (again, see Fig. 2.7), we get $\alpha+2\beta = 1$ since we have defined branch lengths in expected substitutions

per site. This gives us the following rates for transitions and transversions:

$$\begin{aligned}\alpha &= \frac{R}{R+1} \\ \beta &= \frac{1}{2} \times \frac{1}{R+1}\end{aligned}$$

since those values guarantee us that

$$\underbrace{\left(\frac{R}{R+1}\right)}_{\alpha} + 2\underbrace{\left(\frac{1}{2} \times \frac{1}{R+1}\right)}_{\beta} = 1$$

which is required for the transition/transversion ration to be R and the total rate of change to be 1.

For a branch of length t we then get the following probabilities that a transition or a transversion, respectively, has occurred:

$$\begin{aligned}P\left(\text{transition} \mid t\right) &= \frac{1}{4} - \frac{1}{2}\exp\left(-\frac{2R+1}{R+1}\right) + \frac{1}{4}\exp\left(-\frac{2}{R+1}t\right) \\ P\left(\text{transversion} \mid t\right) &= \frac{1}{2} - \frac{1}{2}\exp\left(-\frac{2}{R+1}t\right)\end{aligned}$$

It is important to note that these equations represent the probability that there is a "net" transition or transversion along a branch of length t – there may in fact have been multiple changes between the start and end. The sequence of changes $A \to G \to T$ thus counts as *one* change, from A to T. The probability of no change is

$$\begin{aligned}P\left(\text{neither} \mid t\right) &= 1 - P\left(\text{transition} \mid t\right) - P\left(\text{transversion} \mid t\right) \\ &= \frac{1}{4} + \frac{1}{4}\exp\left(-\frac{2}{R+1}t\right) + \frac{1}{2}\exp\left(-\frac{2R-1}{R+1}t\right).\end{aligned}$$

2.2 Evolutionary Algorithms

Evolutionary algorithms (EAs) is a class of heuristic optimization algorithms, i.e. methods that iteratively try to improve a set of candidate solutions according to some given criteria. EAs are inspired by the principles of evolution through natural selection and molecular genetics. As other heuristic approaches, an EA does not guarantee that an optimal solution is ever found,

but EAs are often used successfully to search large, unpredictable problem spaces partly because they make very few assumptions about the problem being optimized [11].

It should be noted that these *artificial* evolution methods are not intrinsically better for studying *biological* evolution than other heuristic optimization methods – it is simply a good way to generate solutions to optimization problems. In Joseph Felsenstein's words [10]:

> *Genetic algorithms are not inherently more suited to analysis of genetics and evolution than they are to design of bridges.*

A criticism against artificial evolution is that it contains elements of randomness and lacks formal proofs of convergence as other model-based, optimization techniques does [11]. When presented with a good result, it can be difficult to reason about *how* the result was achieved.

2.2.1 Artificial Evolution

Biological evolution is an open-ended process without a predefined goal. In evolutionary algorithms, however, the evolution is *guided* by defining a function that assigns a "fitness value" to an "individual" (a solution) which represents its ability to solve the problem at hand. The selection in an EA is thus *artificial* in contrast to the natural selection in Darwinian evolution.

Evolutionary Algorithms works by mimicking biological evolution and "evolve" increasingly better solutions to optimization problems. This artificial evolution is based on the same pillars as natural evolution: (1) Maintenance of a population, (2) creation of diversity among the individuals, (3) a selection mechanism, and (4) genetic inheritance [11]. EAs thus "evolve" candidate solutions by using analogues to the biological processes of mutation, recombination, and selection.

Each candidate solution is treated as an individual in an iterative "EA cycle" where they are first evaluated by a *fitness function* before the best individuals – as decided by a *selection operator* – are allowed to contribute offspring to the next generation. The offspring are generated by a *recombination operator* which combines the genomes of two individuals to generate offspring. The genomes of the children are then mutated by a *mutation operator* with a probability given by a *mutation rate* which is usually fixed. An illustration of the main steps in such a typical EA is shown in Fig. 2.8 on the following page.

There are several types of EAs. The main difference is in their choice of genetic representations and operators. In this thesis we are mainly concerned with the most common type of EAs, *genetic algorithms*.

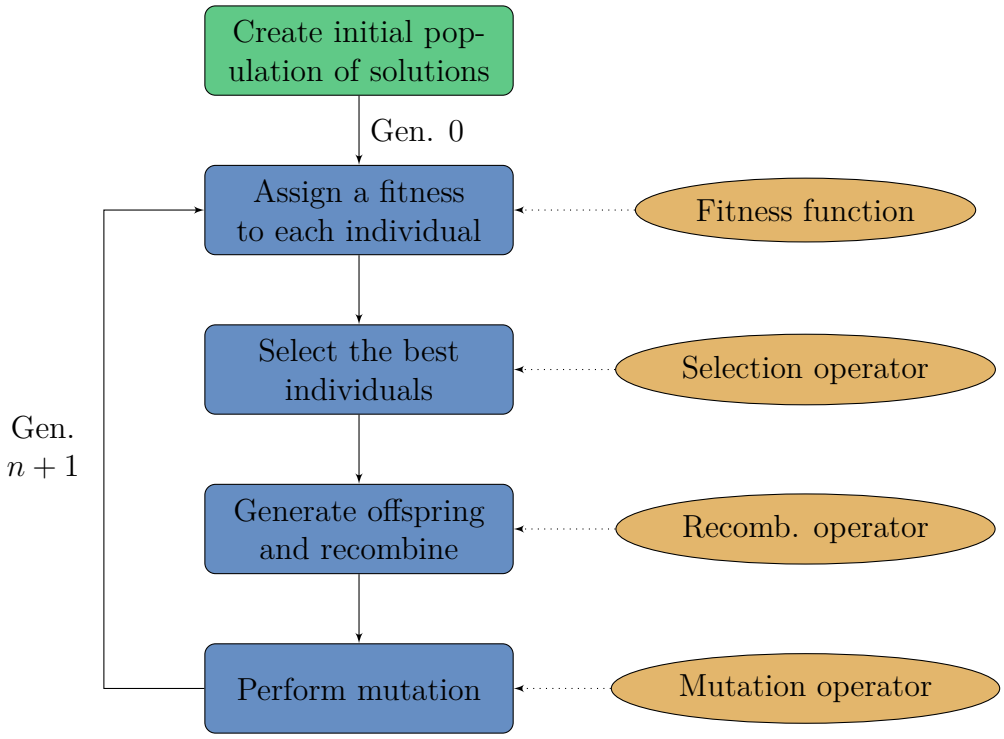


Fig. 2.8: Overview of an evolutionary algorithm. The process is usually run either until a predefined number of generations have been generated or until a solution is found which fulfils a predefined criteria.

2.2.2 Genetic Algorithms

Genetic algorithms (GAs) simulate genetic systems by evolving a population of "genomes" whose fitness is evaluated for each generation. The individuals with the highest fitness have a higher probability of being selected to create offspring for a new generation. When producing the next generation, two and two genomes are recombined to produce offspring until the population of the next generation has become the same size as the current generation. The recombination that is done in genetic algorithms means that good portions of different solutions can be combined. This sets them apart from other heuristic search algorithms [23].

The *genotype* is a string of symbols from a finite alphabet. To calculate the fitness of an individual this data is translated into a *phenotype* which is then evaluated by the fitness function. The genotype–phenotype mapping can be as simple or complex as the designer of the EA system wants it to be. An important feature of any EA is that properties of the parents should – with a high likelihood – be conserved when producing offspring [26]. This means that a great amount of consideration needs to go into the design of the recombination operator.

Two important concepts for a genetic algorithm is *locality* and *heritability*. A genotype–phenotype mapping exhibits locality if small changes in the genotype lead to small changes in the phenotype. This is important for any genetic algorithm to succeed. If a small change in the genotype leads to a disproportionate change in the corresponding phenotype the GA will not be able to find optima by small adjustments as these changes will tend to move the phenotype too far away.

The mutation and recombination performed on the genome in a genetic algorithm is done by *genetic operators*. These operators are applied with a given probability for each generation.

The *mutation operator* is applied to individual genomes and usually results in a small component being changed [7]. The *mutation rate* p_m, $0 \leq p_m \leq 1$ is the probability of a mutation occurring. This probability can be on the "gene" level, meaning that p_m is the probability that each position of the genome is changed, or at the genome level. If defined at the genome level, p_m represents the probability of *some* change occurring to the genome. This change is then done by the mutation operator which will make a (random) change to the genome. The mutation rate is usually quite low – often as low as 0.001 when operating on the gene level of long genomes [26].

The *recombination operator* (or *crossover operator*) is applied to pairs of genomes and results in one or more new genotypes that is a combination of the two "parent" genotypes. The *recombination probability* is traditionally

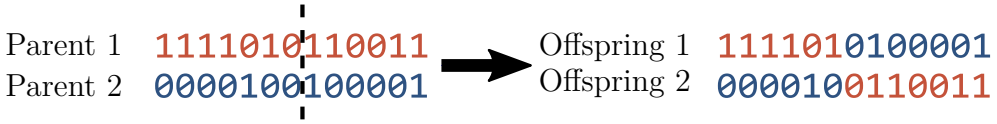


Fig. 2.9: One way to do recombination in a genetic algorithm. A random "recombination point" is chosen where the parents' genomes are cut. The resulting offspring inherits parts of each parent's genome.

denoted p_c and is used to determine whether the offspring of two parent genotypes should be produced by recombination or by directly copying the genotypes to the next generation. Three common recombination schemes are [7, 11]:

n-point crossover Finding n points along the length of the genomes where they are cut. An offspring receives alternating parts of the parents' genomes. Single-point ($n = 1$) crossover is illustrated in Fig. 2.9 .

Cut and splice The parent genomes are cut at a random location (not necessarily the same location) and the parts are then combined to generate offspring. The resulting genomes will in most cases have different lengths.

Uniform crossover The genetic content is exchanged at n randomly chosen positions.

The *selection operator* is responsible for selecting an individual from the population given their fitness values. A good individual should have a better probability of being selected than a poor individual, but the magnitude of this difference varies among different selection strategies. Selection strategies can be *global* or *local*. With a global selection strategy each individual is assigned a fitness and the individual competes with all other individuals, while in local selection strategies subgroups of the population compete among themselves [7]. Examples of global selection strategies are

- *fitness-proportionate selection* where an individual's probability of being selected is directly proportionate to its fitness;
- *rank selection,* where an individual's fitness among n individuals is n for the best individual, $n - 1$ for the second best, and so on;
- and *Boltzmann selection* where fitness scores are scaled in such a way that selection pressure increases for each generation, making it less likely that a low-scoring individual is selected [7].

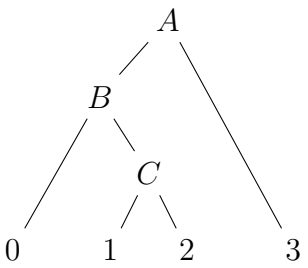


Fig. 2.10: Encoding this example tree into prefix notation yields the sequence $A, B, 0, C, 1, 2, 3$.

A related concept is *elitism*. When using elitism, the n best individuals are directly copied to the next generation without having to compete with others. This ensures that the maximum fitness cannot decrease during an EA run and that good solutions are never lost.

2.2.3 Representing Trees

Two common ways of coding the topology of a tree is prefix notation and Prüfer numbers.

A *prefix representation* of a tree lists the nodes in the sequence they are visited by a preorder traversal of the tree. Fig. 2.10 shows an example tree with four leaf nodes and three internal nodes. A preorder traversal of this tree would first visit the root, then traverse the left subtree followed by the right subtree. Applying this recursively to the example tree in the figure, and noting when nodes are visited, results in the sequence $A, B, 0, C, 1, 2, 3$.

To build a tree from a sequence we need to know the arity of each of the nodes (i.e. how many child nodes are connected to them). In a bifurcating phylogenetic tree all nodes have an arity of two except for leaf nodes which are terminal nodes and thus have an arity of zero. By keeping a stack of "missing arity" the tree can be built by adding each node as a child to the previous node that has an available slot. For the example tree we would start with the node A which is an internal node and therefore starts with an arity of -2 (two missing children). Adding B as its left child means that its arity is -1. This is pushed to the stack and we shift our focus to B which is missing two children. Node 0 will fill one slot and C the next, giving B an arity of zero. Adding nodes 1 and 2 gives C a zero arity. We then pop values of the "arity stack" until we find the first non-zero value, which will be -1 belonging to A. Adding node 3 fills the last slot and concludes the tree building.

The *Prüfer sequence* (also referred to as Prüfer numbers) is a compact

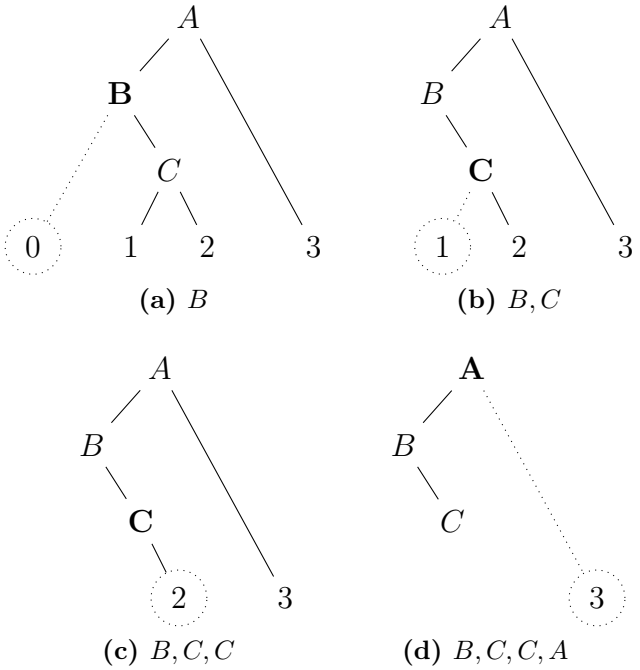


Fig. 2.11: The four first steps of encoding an example tree with seven nodes into its Prüfer sequence representation. The last step (not shown) is removing A and appending B to the sequence. The final Prüfer sequence, uniquely representing this tree, is thus B, C, C, A, B.

encoding for trees. It allows a tree of n nodes to be uniquely represented by a string of $n - 2$ node labels.

To come up with the Prüfer sequence for a tree there first has to be an ordering of the node labels, e.g. $0, 1, 2, \ldots, A, B, C, \ldots$. The next label in the sequence is then found by removing the "smallest" label and appending *the node it is connected to* to the sequence. For the example tree shown in Fig. 2.11 the first step would be to remove node 0 and add B to the sequence. The next nodes would be node 1 and node 2, which are connected to node C, yielding the sequence B, C, C after three steps. Node 3 is then removed and A added to the sequence before, in the last step, node A is removed and B is appended to the Prüfer sequence. The two last nodes are not necessary to include in the sequence. The final result is thus B, C, C, A, B.

To go from a Prüfer sequence to a tree one first has to find the degree of each node. This is simply done by counting the occurrences of its label in the sequence and adding one, giving node 0 degree $0 + 1 = 1$, while node A has degree $1 + 1 = 2$. One can then do the reverse of the encoding process: For each node in the sequence, add an edge from that node to the first node

with degree 1 and reduce both nodes' recorded degree. This is done until you have only two nodes left which should be connected.

Several people have voiced concerns about the use of Prüfer numbers as a genotype in genetic algorithms due to poor locality and heritability in many cases, e.g. [31] and [12]. Rothlauf's conclusion in a 2000 study of GAs using Prüfer sequence encoding was that their performance for trees were poor [31]. In a 2001 study together with Gottlieb, Julstrom, and Raidl [12] where they evolved spanning trees, the conclusion is even bolder:

> *Our conclusion is definite: Prüfer numbers cause poor performance in evolutionary algorithms and should be avoided.*

However, other researchers have used Prüfer sequences with good results. In [17] Hassan et al. created a multi-objective evolutionary algorithm shown to perform well compared to two established systems[4]. Hassan et al. concluded that their proposed Prüfer sequence based method showed promise and should be investigated further. They did not, however, explain what – if anything – they did to avoid the problems outlined in [12].

2.3 Maximum Likelihood

Maximum Likelihood is a Bayesian learning method. Like other Bayesian methods it is based on Bayes' theorem, which provides a way to calculate the *posterior* probability $P(h \mid D)$ from (1) the *prior* probability of a hypothesis, $P(h)$; (2) the prior probability that the data D will be observed, $P(D)$; and (3) the probability of observing D given that h is true [26]:

$$P(h \mid D) = \frac{P(D \mid h)P(h)}{P(D)} \tag{2.2}$$

The most probable hypothesis h from a set of candidate hypotheses H given observed data D is called a *maximum a posteri* (MAP) hypothesis. To find this hypothesis we can use Bayes theorem (Equation (2.2)) to calculate the probability of each hypothesis given the observations and pick the most likely candidate:

[4] DNAPARS and DNAML from the PHYLIP package

$$
\begin{aligned}
h_{MAP} &\equiv \underset{h \in H}{\operatorname{argmax}} P(h \mid D) \\
&= \underset{h \in H}{\operatorname{argmax}} \frac{P(D \mid h) P(h)}{P(D)} && \text{(Bayes theorem)} \\
&= \underset{h \in H}{\operatorname{argmax}} P(D \mid h) P(h) && (1/P(D) \text{ is constant}) \qquad (2.3)
\end{aligned}
$$

If we further assume that each hypothesis in H has the same probability of being true we can simplify Equation (2.3) to only look at $P(D \mid h)$, i.e. the likelihood of observing D given hypothesis h. The hypothesis that maximizes this quantity is called the *Maximum likelihood* (ML) hypothesis, h_{ML}:

$$
h_{ML} = \underset{h \in H}{\operatorname{argmax}} P(D \mid h)
$$

This means that we make no assumptions of prior probabilities over the space of hypotheses, meaning that no subjective evaluation of hypothesis priors is done while it still provides a good approximation to MAP learning when the data set is large [34].

It has been shown that ML estimates have several good properties including converging to the correct value and having the smallest possible variance as the amount of data grows large [10].

2.4 File Formats

To work with phylogenetic inference we need to represent trees and the sequence data (DNA) that belongs to the leaf nodes of these trees. Most file formats in the field of bioinformatics are text-based and it is usually easy to convert between different formats. Some common formats relevant for this thesis are presented here.

2.4.1 Phylogenetic Trees

The most common file format for representing a phylogenetic tree is the Newick format [10]. The Newick format describes trees with nested expressions using parentheses. There has never been a formal specification, but a description of the file format is available from the PHYLIP page at the Felsenstein/Kuhner lab web page[5]. In the Newick format each internal node

[5] http://evolution.genetics.washington.edu/phylip/newicktree.html. Accessed: 2011-02-12. (Archived by WebCite at http://www.webcitation.org/5ycvWdsxO)

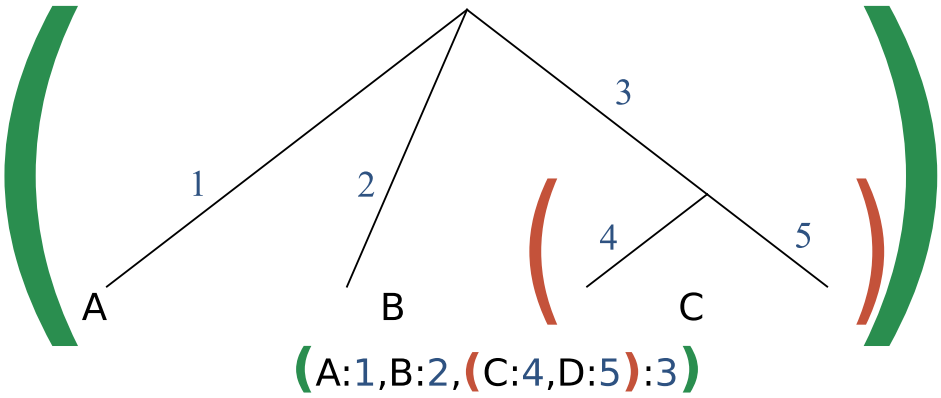


Fig. 2.12: A tree represented in the Newick format. There are two parentheses pairs corresponding to the two internal nodes: the root node – coloured in green, and the subtree with leaves C and D, coloured in red.

– including the root – is represented by a pair of parentheses enclosing the subtrees that are immediately descended from that internal node. Leaf nodes are named, and after both leaf nodes and internal branch points an optional branch length can be specified (separated by a colon character). A tree with a root and two children, A and B, with no branch lengths, would thus be represented in the Newick format as (A, B); – the semicolon functions as an end marker.

The example tree shown in Fig. 2.12 will be represented as

$$(A : 1, B : 2, (C : 4, D : 5) : 3);$$

in the Newick format. The node labels are A–D and all branch lengths are specified. For example, the "C,D" subtree is connected by a branch of length 3.

While Newick has remained the de facto standard for many years, several new XML-based file formats have appeared. One promising XML-based format for phylogenetic trees is PhyloXML as described in [15].

2.4.2 Sequence Data

Representing DNA sequence data is easy: simply list the observed base at each site, i.e. A, C, G, or T. However, there is often uncertainty in the data from DNA sequencing and the file formats allow for uncertainty in the sequences by using a wider range of symbols. These symbols allow a site to be marked as containing a *purine* (A or G), a *pyrimidine* (C or T), a base that is *"not A"* and so on. Most file formats for sequence data are text-based and it is usually easy to convert between them [14].

```
> Species1
ACGTGGA--CGGATTGCATCGTGATTGCFFTAACGTA
AATCTCATTCTATCACACATCATTTACTTTCATTTTT
> Species2
AGAGATTGCAAGAATGCATCGTACCCAATGCATCGTA
CATTCTATTATTATTATTATCACAGGAGGATTACTAG
```

Fig. 2.13: An example FASTA file with two sequences.

```
    2    74
Species1    ACGTGGA--C GGATTGCATC GTGATTGCFF
Species2    AGAGATTGCA AGAATGCATC GTACCCAATG

            TAACGTAAAT CTCATTCTAT CACACATCAT
            CATCGTACAT TCTATTATTA TTATTATCAC

            TTACTTTCAT TTTT
            AGGAGGATTA CTAG
```

Fig. 2.14: An example interleaved PHYLIP file with two sequences.

A simple format in common use is the FASTA format from a software package of the same name. This is a text-based format for representing either nucleotide sequences or amino acid sequences [14]. Bases (or amino acids) are represented using single-letter codes. The sequences can be named. A description of the FASTA format is available from the National Center for Biotechnology Information at `http://www.ncbi.nlm.nih.gov/BLAST/fasta.shtml`[6]. An example of a FASTA is shown in Fig. 2.13 .

The PHYLIP format is another widely used format from a free package of programs with the same name. The format is documented at `http://evolution.genetics.washington.edu/phylip/doc/sequence.html`[7]. PHYLIP files usually list sequences *interleaved*. This is illustrated in Fig. 2.14 which shows two sequences split up in alternating lines. The first line specifies the number of sequences and their length.

[6] Accessed: 2011-05-13. (Archived at http://www.webcitation.org/5ycwAOEt1)
[7] Accessed: 2011-05-13. (Archived at http://www.webcitation.org/5yySI0n5J)

2.5 Parallel Execution

The terms *parallel* and *concurrent* are often synonymous, but in computer programs they describe different concepts. The following definition of a parallel program is due to Simon Marlow [24]:

> *A parallel program is one that uses a multiplicity of computational hardware (e.g. multiple processor cores) in order to perform computation more quickly. Different parts of the computation are delegated to different processors that execute at the same time (in parallel), so that results may be delivered earlier than if the computation had been performed sequentially.*

This is contrasted with concurrency, which is a technique of building programs to have multiple threads of control. Such a program can be executed on several processors, meaning that the effects of the threads will be interleaved; or on a single processor, where only one thread will be running at any given time.

Evolutionary algorithms lend themselves easily to parallel computing [10]. In a typical genetic algorithm with a simple genome, mutation and recombination are cheap operations while fitness evaluation requires much more computation. Given that fitness evaluation of an individual is independent of other individuals, these computations can be distributed among several computers and computed in parallel [18]. Fig. 2.15 on the next page shows a master/slave setup where a master node passes the genomes to several slave nodes for fitness evaluation.

2.6 Related Work

Evolutionary algorithms have been used for phylogenetic inference at least since 1996 when Hideo Matsuda [25] used a genetic algorithm for phylogenetic inference based on amino acid sequences. He concluded that this approach produced results comparable to the conventional tree construction methods in use. The `GAML` program written in 1998 by Paul O. Lewis was inspired by Matsuda. An overview of `GAML` is presented in the following section. Another modern implementation of a genetic algorithm for phylogenetic inference published by Cotta & Moscato in 2002 is also reviewed.

2.6.1 GAML (1998)

GAML (genetic algorithm for maximum likelihood phylogeny inference) [23] is a program written by Paul O. Lewis in 1998. The program uses the

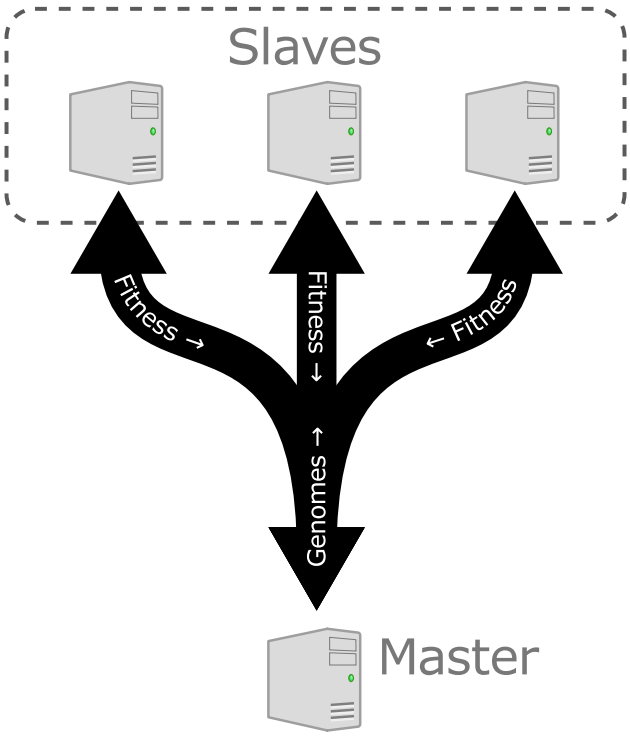


Fig. 2.15: The EA master node can distribute the genomes to other machines for evaluation. The slave nodes read a genome and return a fitness value.

maximum likelihood criterion for evaluating trees and implements the HKY substitution model – an extension of the Kimura model that allow different prior probabilities for the four possible bases.

In `GAML`, both tree topologies, branch lengths, and the value for the transition/transversion ratio parameter for the HKY model are evolved. By evolving branch lengths as a part of each tree, `GAML` does not need to optimize branch lengths as part of the fitness evaluation. The author states that considerable time is saved in this step over other methods.

Lewis ran `GAML` on a 55-taxon data set until no improvement was seen for 2000 generations. This was done three times and the trees were compared with another heuristic search done by `PAUP* 4.0` – a widely used program for phylogenetic inference. One of the three runs produced the same tree as found by `PAUP*`, the other two had only slightly worse likelihood. GAML used between 11.3–16.3 hours while `PAUP*` used 783.2 hours. Further optimizing branch lengths on the trees produced by `GAML` had little effect on the trees' likelihoods.

Lewis concludes that using genetic algorithms for phylogeny inference holds much promise. He also highlights the fact that GAs are easy to parallelize as an advantage of this method:

> *Genetic algorithms (and especially parallel implementations of GAs) offer the potential for inferring maximum-likelihood trees for large data sets involving hundreds of sequences.*

2.6.2 Cotta & Moscato (2002)

Cotta and Moscato [6] tested various EA methods for phylogeny inference. They used a distance based approach where only a matrix with pairwise distances is used, as explained in Section 2.1.4 on page 11.

To evaluate a tree they build an *inferred* distance matrix $\hat{M}$ from their tree and compare this to the *observed* distance matrix M that was built from the genetic data. They measure the quality of a tree by constraining edge weights to $\hat{M}_{ij} \geq M_{ij}$ then compare the total weight of the tree with the sum of distances in M_{ij}. The best tree is thus the tree that minimizes this sum of edge weights under this constraint.

One EA using the direct approach and two decoder-based EAs were implemented and compared. They found that the direct encoding performed better than decoder-based EAs for a small problem set (20 species) but that it did not scale well with problem size. For a larger problem set (34 species) an indirect approach had the best performance and needed less computation than other methods. The decoder they used was "greedy": it assessed

the quality of the tree it was building while nodes were added. The most promising of the possible subtrees was always picked.

Their conclusion is that directly evolving phylogenetic trees gives better results than indirect approaches except in the case of the "greedy" decoder which produced optimal or near-optimal solutions with lower computational cost than all other methods.

Chapter 3

Methods

This chapter looks at the methods used to calculate a tree's likelihood. A discussion of sources of error is also included, and the last section looks at GA operators that can be used in an EA to evolve phylogenetic trees.

3.1 Calculating the Likelihood of a Tree

A phylogenetic tree's fitness – as seen from the evolutionary algorithm – is its *likelihood* in light of the genetic data and a model of evolution. The following explanation is based on the accounts in Joseph Felsenstein's book on inference of phylogenies [10] and the chapter on likelihood calculation in molecular phylogenetics in *Mathematics of evolution and phylogeny* [3].

To calculate the likelihood of a given tree we use a model of DNA evolution that allows us to compute the transition probability $P(j \mid i, t)$ – the probability that state j will exist at the end of a branch of length t if the state at the start of the branch is i. The branch length t is measured in *expected substitutions per site* rather than time, i.e. along a branch of length 0.5 we expect to see changes happen to half of the sites. (Refer back to Fig. 2.6 on page 12 to see an illustration of this concept.)

An example tree is shown in Fig. 3.1 on the following page. In this tree the evolutionary relationship between four "species" is shown. The actual species, which will always be leaf nodes in a tree, are called *operational taxonomical units* (OTUs) while the interior nodes are called *hypothetical taxonomical units* (HTUs). The latter are *inferred* ancestral species of which we have no observed data. In this simple example tree we only look at one *site*. We have observed the states A, C, C, and G for our OTUs. The hypothetical states of the HTUs are x, y, and z.

When calculating the likelihood of a whole phylogenetic tree we assume

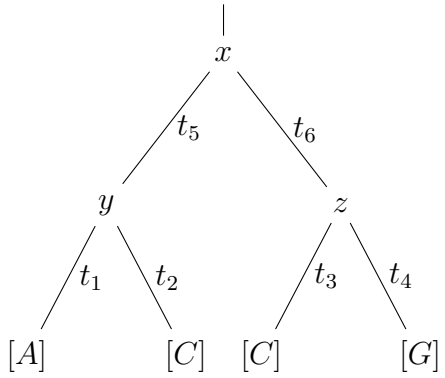


Fig. 3.1: Our hypothetical phylogenetic tree. The values A, C, C, and G have been observed for the leaf nodes.

the following:

1. Evolution in different sites is independent.

2. Evolution in different lineages is independent.

The first assumption lets us decompose the likelihood of a tree given the data, $P(D \mid \mathbb{T})$, into a product of n terms – one for each site:

$$L = P(D \mid \mathbb{T}) = \prod_{i=1}^{n} P(D_i \mid \mathbb{T})$$

where D_i is the data at site i. The likelihood of a tree with observations of one state is then the sum of the probabilities of all possible combinations of states which might have existed at the m interior nodes $n_1 \ldots n_m$:

$$\begin{aligned} P(D_i \mid \mathbb{T}) &= \sum_{n_1} \sum_{n_2} \cdots \sum_{n_m} P(s_1, s_2, \ldots, s_n, n_1, n_2, \ldots n_m \mid \mathbb{T}) \quad (3.1) \\ &= \quad P(A, C, C, G,\ A, A, A \mid \mathbb{T}) \\ &\quad + P(A, C, C, G,\ A, A, C \mid \mathbb{T}) \\ &\quad + \ldots \\ &\quad + P(A, C, C, G,\ T, T, T \mid \mathbb{T}) \end{aligned}$$

where $s_1, s_2, \ldots, s_n$ are the leaf nodes' states at that one site (A, C, C, G for our example tree). Each summation runs over the four possible states A, C, G, and T.

The second assumption we made – that different lineages evolve independently – allows us to decompose the probability on the right side of Equation (3.1) into a product of terms. For the example tree we thus get:

$$\begin{aligned} P(A,C,C,G,x,y,z \mid \mathbb{T}) &= P(x) \times P(y \mid x, t_5) \\ &\quad \times P(A \mid y, t_1) \times P(C \mid y, t_2) \\ &\quad \times P(z \mid x, t_6) \times P(C \mid z, t_3) \\ &\quad \times P(G \mid z, t_4) \end{aligned} \tag{3.2}$$

The probabilities above are given by the transition probability of our model of evolution. Assuming that evolution has gone on for a very long time – with the state change probabilities given by our evolutionary model – it is reasonable to use $P(x)$ as the equilibrium probability of state x under that model. The other probabilities in Equation (3.2) are derived from the chosen model of DNA evolution.

It is too inefficient to use Equation (3.2) directly because the number of terms we need to sum rises exponentially with the number of species.[1] A pruning method can be used to make the computation feasible: By moving the summation signs as far to the right as possible we get

$$\begin{aligned} P(D_i \mid \mathbb{T}) = & \sum_x P(x) \\ & \times \left(\sum_y P(y \mid x, t_5)\, P(A \mid y, t_1)\, P(C \mid y, t_2) \right) \\ & \times \left(\sum_z P(z \mid x, t_0)\, P(C \mid z, t_3)\, P(G \mid z, t_4) \right). \end{aligned} \tag{3.3}$$

To compute this we work our way up the tree by calculating the *conditional likelihoods* of subtrees: the probability of what is observed at a node k and down given that k has the state s – denoted $L_k^{(i)}(s)$. This computation is done for each site i.

3.1.1 A Recursive Algorithm for Computing $L_k^{(i)}(s)$

We can use an algorithm that "pushes" information up the tree to compute Equation (3.3). We make use of the subtree likelihoods $L_k^{(i)}(s)$. The term

$$P(C \mid z, t_3)\, P(G|z, t_4)$$

[1] For a tree with n species, there are $n-1$ interior nodes each of which has one of four states, meaning that we need to sum 4^{n-1} terms.

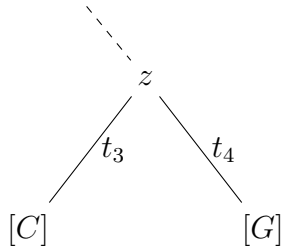


Fig. 3.2: The right subtree of the example tree.

is such a quantity for the left subtree (shown in Fig. 3.2) of the example tree. This can be read as "the probability of everything seen at and below a node with state z". There will be four such quantities – one for each possible value z can take (A, C, G or T). Once these four values have been computed they need not be computed again if we find the same states in the nodes below z for a different site. This dynamic programming approach greatly reduces the number of calculations that needs to be done.

The algorithm is recursive and computes $L_k^{(i)}(s)$ at each node from the same values from the nodes immediately below it:

$$L_k^{(i)}(s) = \left(\sum_x P(x \mid s, t_\ell)\, L_\ell^{(i)}(x)\right)\left(\sum_y P(y \mid s, t_m)\, L_m^{(i)}(y)\right) \tag{3.4}$$

where ℓ and m are the two nodes directly below k.[2] This can be read as "the probability of everything at or below a node – given that this node k has state s – is the product of the probability of the events of both the descendant lineages".

At the leaf nodes we have an observation v of the actual state (A, C, G, or T in the sequence data), so we get

$$L_{\text{leaf}}^{(i)}(s) = \begin{cases} 1 & \text{if } s = v \\ 0 & \text{otherwise} \end{cases}.$$

These values will act as the *base cases* of the recursive algorithm. A pseudo-code implementation of this algorithm is shown as Algorithm 3.1 on the facing page.

The end result of the algorithm for a site i is the average of $\pi_s L_{root}^{(i)}(s)$ – the values at the topmost node in the tree weighted by their prior probabilities:

[2] This is specific for a bifurcating tree but can easily be extended to an n-furcating tree by adding more factors on the right side of equation 3.4.

Algorithm 3.1 An algorithm for calculating the likelihood of a node, $L_k(s)$, as described in equation 3.4 on the facing page.

```
function L(node):
    # base case: we have already found
    # likelihoods for the node
    if node.likelihoods:
        return node.likelihoods

    for s in ['A', 'C', 'G', 'T']:
        xs = ys = 0.0
        for x in ['A', 'C', 'G', 'T']:
            xs += P(s, x, t)*L(node.l_child)[x]
            ys += P(s, x, t)*L(node.r_child)[x]

        node.likelihoods[s] = xs*ys

    return node.likelihoods
```

$$L^{(i)} = \sum_s \left(\pi_s L^{(i)}_{root}(s) \right)$$

The probability of the entire tree is then the product of the probabilities for each site:

$$P(tree) = \prod_i L^{(i)} \tag{3.5}$$

In practice, the likelihoods will be very low, so Equation (3.5) is usually implemented as summation of logarithms instead of multiplying the likelihoods directly.

3.1.2 Evaluation of Computational Cost

For each site we have to calculate the result of Equation (3.3) $n-1$ times, once for every internal node (HTU). Each computation requires four calculations – one for each possible base, and each of these are the product of two terms of four products (since we have four bases).

If we have a tree with n leaves and sequences of length p, the total calculation is proportional to $p(n-1)4^2$. This is an upper bound. As mentioned,

we can use memoization to directly look up the likelihood if the same calculation has been done previously.

3.2 Unrootedness

As mentioned in Section 2.1.3 on page 8 most algorithms for phylogeny inference computes an unrooted tree which only shows the relatedness among the leaf nodes. This is also the case for the ML method outlined in this chapter. There is no information about the placement of the root of the tree [8]. The trees are simply drawn in a way that is familiar to most people. Trees in the Newick format which is the most common file format for phylogenetic trees will always have a root, but it is important to keep in mind that any of the internal nodes can be the *true* root [10].

3.3 Sources of Error

Inferring a phylogeny from sequence data is a probabilistic process and there are many sources of errors. The most important are listed below.

- That evolution in different sites is independent, as assumed by the maximum likelihood method outlined in Section 3.1 is quite unrealistic. It is know that the substitution processes at different sites in a sequence often are not independent [3, 20].

- There can be errors in the DNA sequence data used. Misdentification during sequencing happens, but this is rapidly improving and the rate is currently sometimes as low as 10^{-4} per base [35].

- The alignment process is a probabilistic process and the most probable alignment need not reflect the true mutations that has happened.

- The models of evolution used in the maximum likelihood approach are probabilistic models and will never be completely accurate.

There have also been some concern about using the (often illusory) maximum-likelihood point as an optimality criterion. Steel, in [36], demonstrated that a phylogenetic tree does not always have a unique maximum likelihood point. However, a study by Rogers and Swofford [30] showed that this is not a general phenomenon and showed, through simulations, that maximum-likelihood is still a good optimality criterion. Rogers and Swofford conclude:

Our results thus provide reassurance that the value of maximum likelihood as a tree selection criterion is not compromised by the presence of multiple local maxima—the best estimates of the true tree are not likely to have them. This result holds true even when an incorrect substitution model is used for tree selection.

3.4 GA Operators for Phylogenetic Trees

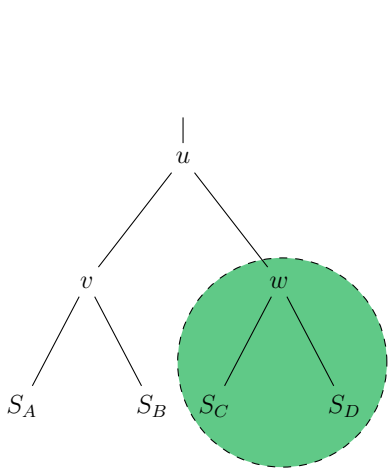
The field of *genetic programming* (GP) has produced mutation and recombination operators suitable for evolving trees. Trees in GP usually represent computer programs and not all of the operators suitable for these trees work for phylogenetic trees due to the strict restrictions on their form [6]. A recombination operator that works for phylogenetic trees is the *Prune-Delete-Graft* (PDG) operator. PDG works by selecting a random node from one of the parent trees as root in the subtree under it. The leaf nodes that appear in this subtree is then pruned from the other parent. Last, a random branch in the pruned tree is selected to which the random subtree from the other parent is attached. This method is illustrated in Fig. 3.3 on the following page. This method is quite simple will always produce a valid tree.

Several methods are used to implement a mutation operator. Cotta and Moscato [6] lists the following mutation operators for tree topologies:

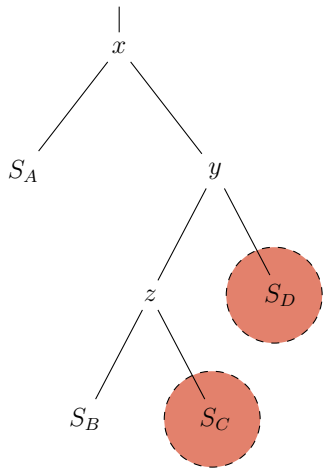
SWAP**:** two random leaf nodes swap positions

Nearest Neighbour Interchange**:** switch two branches that share a neighbour branch

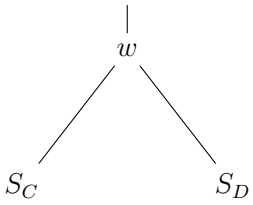
SCRAMBLE**:** a random subtree is chosen and its topology is rearranged at random



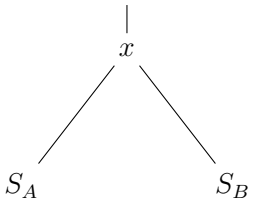
(a) Parent A



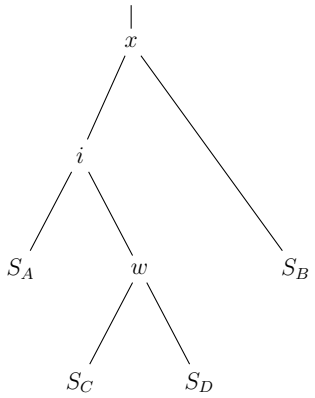
(b) Parent B



(c) The selected subtree from A



(d) Parent B after pruning



(e) The resulting "offspring" tree.

Fig. 3.3: Recombination of two parent trees, A and B with *Prune-Graft-Delete* is done by finding a random subtree of A. In this example the subtree under the w node (green circle) is chosen. The leaf nodes that appear in this subtree are pruned from parent B (red circles), resulting in the tree shown in Fig. d. The selected subtree (Fig. c) is then inserted at a random point – here between x and S_A. resulting in the tree shown in Fig e.

Chapter 4

Implementation

4.1 Overview

An evolutionary algorithm (EA) system for evolving phylogenetic trees for a given set of aligned DNA sequences using a maximum likelihood (ML) approach has been implemented. The system is highly modular and it is easy to change the operators (mutation and crossover) for the genetic algorithm and the evolution model for the ML evaluation of trees.

The two main parts of the system and their sub systems are:

- An **evolutionary algorithm program** responsible for maintaining a population of trees, passing these to a program for evaluating their fitness and then creating the next generation by recombining the best trees (see Fig. 2.8 on page 16). This program makes use of the following components to achieve this:
  - ▷ A **selection operator** using one of a set of selection strategies.
  - ▷ A **mutation operator** for trees.
  - ▷ A **recombination operator** for trees.
- An **evaluation program** which, given a tree, calculates the tree's fitness. This program uses the ML approach as outlined in Section 3.1 on page 29. This program has only one swappable component:
  - ▷ An **evolution model** which is used to calculate the probabilities of changes in the tree.

The code is written in C++ and modularity is achieved using object-oriented design principles. By using *templates*, any genome representation can be used

as long as GA operators for this format are implemented. This means that a genome can be a text string, an array of integers or any other data structure, as long as, e.g. the fitness function that is used knows how to interpret this genome.

4.2 Implementation details

4.2.1 EA System

The EA system runs the "EA cycle" (see Fig. 2.8 on page 16). This system is quite simple and is responsible for using the operators to create new generations until a supplied criterion (usually a specified number of generations or convergence) is fulfilled.

4.2.2 Tree Evaluator

The tree evaluator reads in a tree genome and build the corresponding data structure. The likelihood algorithm is then run on the root node which recursively computes the likelihood. As mentioned in Section 3.1.1 on page 31 the values for equation (3.4) can be cached and need only be recalculated every time the branch length they were computed for changes. This memoization saves a lot of time since at most 16 values need be computed for each node pair ($A \to A, A \to C, \ldots, T \to T$). Testing this on a tree with seven sequences of 1000 bases showed that the calculations took only 1/43 of the time compared to not using memoization.[1]

4.2.3 Data Structures

A few data structures (C++ classes) constitute the core of the EA system. They are described below.

- `PhyloTreeNode`

  Represents a node in a phylogenetic tree. Both internal nodes and leaf nodes (species) are of this class. A node has two children (leaves) – one "left" and one "right" – or no children. It has one parent (except for the root node). All nodes also contain a cache for likelihood values together with the branch length the values were cached for so that the cache can be invalidated when needed. The most important member functions are:

[1] Without memoization: 2.737 s ($\sigma = 0.06114$). With memoization: 0.0634 s ($\sigma = 0.003921$).

- ▷ `vector<vector<double>> likelihood(EvolutionModel*)`

  Calculates likelihood values for each site for each possible base. If there are n sites this will be an $n \times 4$ matrix.

- ▷ `string prefixRepresentation(PhyloTreeNode* node)`

  Returns a string representation of the tree in the prefix format described in Section 2.2.3 on page 19.

- `PhyloTree`

  A helper class for working with trees of `PhyloTreeNode` objects. A `PhyloTree` contains a pointer to the `PhyloTreeNode` object that represents the tree's root and an `EvolutionModel` object. The goal of this class it to make it easy to work with `PhyloTreeNode`s that are linked together to form a tree. The most essential functions are:

  - ▷ `double logLikelihood();`

    Calculates the tree's total likelihood found by multiplying all site likelihoods as explained at the end of Section 3.1.1 on page 31. Since these probabilities will be very low, the sum of the logarithm for each site is returned.

  - ▷ `string newick()`

    Returns a string representation of the tree in the Newick format.

  - ▷ `void buildRandomTree(vector<PhyloTreeNode*> leaves);`

    Builds a random tree with the given leaf nodes. This is done by first building a tree of the internal nodes by recursively adding one or two (with a 50% probability) orphan nodes as children. This is done until all internal nodes are connected. The leaves are then randomly sorted and added as children to the internal nodes that have no children or only one child. This is done to create the first generation of random trees.

  - ▷ `void setEvolutionModel(EvolutionModel* m)`

    Sets the evolution model (see Section 2.1.5 on page 12) to the given object. The evolution model is used to calculate the transition probabilities for the likelihood computation.

  - ▷ `static PhyloTree decodePrefixNotation(vector<PhyloTreeNode*> nodes, string s, EvolutionModel* evModel);`

    This function builds a tree from a prefix representation. This is the genotype→phenotype mapping done in order to evaluate a genome's fitness.

- `EvolutionModel`

  This is an abstract class representing a probabilistic model of evolution. The system includes two concrete subclasses: `JukesCantor` and `Kimura`, representing those models. An `EvolutionModel` only has two functions:

  - ▷ **`double P(char i, char j, double t)`**

    The probability of a change from i to j over a branch of length t (see Equation (2.1) on page 12).

  - ▷ **`double prior(char s)`**

    Returns a base's prior probability, i.e. its equilibrium probability. (For both the Jukes-Cantor and the Kimura model this is simply $\pi_{\{A,C,G,T\}} = 1/4$.)

The EA operators are described by the four abstract classes described below. These abstract classes act as interfaces that are implemented by concrete subclasses. (These classes use C++ templates to make sure that they would work for any genome, not only strings. A `'T'` in the function signatures below can thus be `string` or any other data type.)

- `MutationOp`

  A MutationOp is responsible for mutating a set of genomes. It will in most cases have the mutation probability as a member variable. Its only member function takes a set of genomes to be mutated.

  - ▷ **`void mutate(vector<T>& genomes)`**

    Takes genomes to be mutated as a parameter and (with possibility p_m) mutates them.

- `RecombOp`

  A recombination operator can return one or more children, the EA system will be responsible to select and recombine parents until a new generation has been produced. A recombination operator implementation will usually have p_c as a member variable and simply return the two parents in the cases where no recombination should be performed.

  - ▷ **`vector<T> produceOffspring(T& p1, T& p2)`**

    This function takes two parents and return one or more children.

- `SelectionOp`

A selection operator is an implementation of a selection strategy (see Section 2.2.2 on page 18). Two concrete subclasses of this abstract class has been implemented: `FitnessProportionateSelection` and `RankSelection`. The only member function performs this selection.

  - ▷ `T& select(vector<T>& pop, vector<double> fitness)`
    Takes a set of genotypes together with a set of their corresponding fitness values and returns one individual.

- `FitnessFunc`

  A fitness function is responsible for evaluating a genome's fitness. This usually includes translating the genotype to a phenotype and assessing its fitness. Its only member function takes a set of genomes and returns their fitness values.

  - ▷ `vector<double> fitness(vector<T>& genomes)`
    Takes a list of genomes and returns their fitness values.

4.2.4 Differential Reproduction

In GAML (see Section 2.6.1 on page 25), Lewis let the number of offspring produced be proportional to the parents' rank likelihood score. An analogue to this is also present in this system: By enabling differential reproduction, above-average parents are allowed to produce more offspring. This was done by allowing all parent pairs to produce one offspring plus one extra offspring per parent whose fitness is above average. Two above-average parents will thus have 3 offspring passed on to the next generation.

4.2.5 GA Operators for Phylogenetic Trees

Selection is done by using *rank selection*. Since the likelihood of any tree is very small it is convenient to use the log likelihoods directly in the selection strategy. By using rank selection the *magnitude* of the fitness differences has no effect, only the value's ranking among all other fitness values.

4.2.6 Reading Sequence Data

Sequence data is read from FASTA files (described in Section 2.4 on page 22). The parser is only written to handle the DNA nucleic acid codes (A, C, G, T). The parsing function takes the file name of a FASTA file as an argument and returns a set of `PhyloTreeNode` objects (one per sequence) whose states are the read sequences.

4.2.7 Displaying Trees

The system supports exporting trees in the Newick format. This format (see Section 2.4 on page 22) is the most common format for representing phylogenetic trees. Programs such as `nw_display` from the software package *nw_utils* [22] can be used for visualizing the trees or export them as vector graphics files.

4.2.8 Testing

Invariants in the system are checked at build time with tests written in the Google Test Framework[2]. These tests include:

- calculating the fitness for trees with known fitness values,
- testing that the transition probabilities for pairs of random nodes is within $[0, 1]$, and
- testing that max fitness never decreases from one generation to the next when using elitism.

The memory debugger/profiler Valgrind[3] has been used to make sure there are no memory leaks in the program and to find performance bottlenecks.

4.3 Parallelization

It has been a design goal that the system should be easy to parallelize with *Simdist* [18] – a tool for distributing the work in an evolutionary algorithm between a master node and slave nodes (see Fig. 2.15 on page 26 for an illustration of this concept). Simdist works by passing genomes and the resulting fitness scores between programs through the standard input/output streams. The system has therefore been split into two separate programs; one "master" program which maintains the population and creates new generations and a "slave" evaluator program which assigns a fitness to trees given to it. There will usually be many instances of the evaluator program running – for optimal performance there should be at least as many instances as available processors.

[2] http://code.google.com/p/googletest/
[3] http://valgrind.org/

4.4 Prune-Delete-Graft with Branch Lengths

Extra consideration has been put into the use of the PDG recombination operator on a tree with branch lengths. Because of the way trees are "pruned" and have another subtree "grafted" onto them, information about branch lengths can be lost.

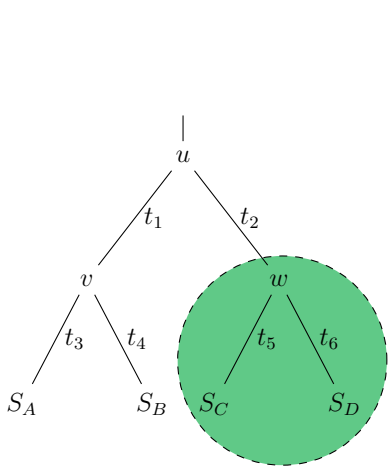
This scenario can be illustrated with an example shown in Fig. 4.1 on the next page: The parent trees A (4.1a) and B (4.1b) are selected for recombination and the highlighted subtree from parent A is to be grafted onto parent B. The selected subtree contains the leaf nodes S_C and S_D, so these nodes should be pruned from B. Removing these leaf nodes from parent B will move S_B up two levels to where y was, resulting in the tree shown in 4.1d. But what should the branch length connecting S_B to the tree (t_α) be? This length can be modified by further search, but choosing the most probable value is important because it will reduce the search time.

Assuming that the parent tree has undergone selection and the branch lengths are reasonably optimized, the distance from x to S_B is optimized and should be preserved, i.e. the length should be set to $t_8 + t_9 + t_{11}$. However, due to a phenomenon called the *lessened node density effect* the branch length that will be recovered in this subset of the parent tree will usually be smaller. The node density effect was described by Venditti, Meade and Pagel in [38] and its effect on evolutionary studies are further investigated in [21]. Because of this effect it is better to pick a shorter branch length, such as t_{11}, because this will quite likely be nearer to the length that subsequently will be recovered by the search. This EA system does this: the branch length from a node whose sibling is pruned is kept as the original distance to its parent node.

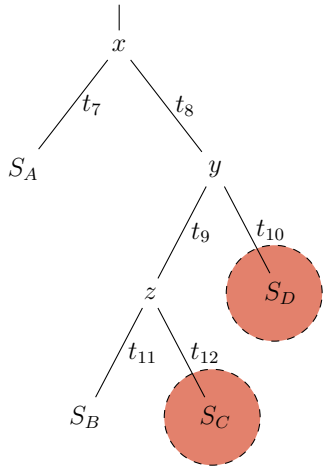
The recombined "offspring", shown in Fig. 4.1e on the following page is also "missing" another branch length – the length of the new branch used to graft the subtree from parent A to the pruned subtree from parent B. In the new tree, the distance that was found from the root to w in Parent A is no longer meaningful, so nothing can be known about this length. It is therefore set to a random value between zero and one.

4.5 Tree Representation

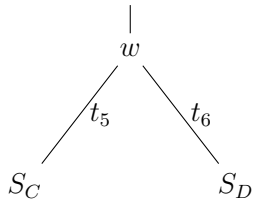
The genome of each "individual" in the EA is a tree represented in prefix notation. The lengths of branches are also specified before the node that is at the endpoint of the branch. An example tree with its corresponding genome is shown in Fig. 4.2 on page 45.



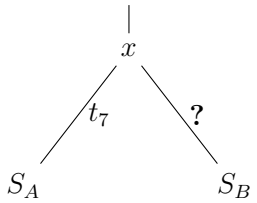
(a) Parent A



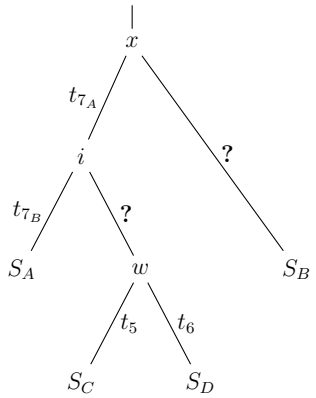
(b) Parent B



(c) The selected subtree from A

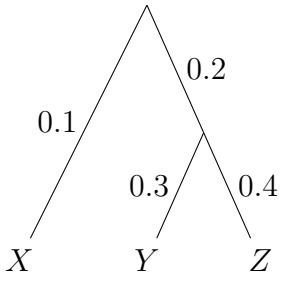


(d) Parent B after pruning



(e) The resulting "offspring" tree.

Fig. 4.1: It is not obvious how to treat branch lengths that are affected during Prune-Graft-Delete recombination. When B is pruned, t_8, t_9 and t_{11} are collapsed to one branch, and when a subtree is "torn off" A, the length of the branch that connected it to its parent is probably no longer meaningful for its new location in a new tree.



(a)

h	0.1	X	0.2	h	0.3	Y	0.4	Z

(b)

Fig. 4.2: Example tree of the three species X, Y and Z (a) and the corresponding "genome" (b). The representation is a preorder traversal of the tree with "h" representing internal nodes (HTUs) and prefixing branch lengths to the nodes.

4.6 Using PHYML to Evaluate Likelihoods

PHYML is a program that is primarily used for phylogenetic inference, but it can also be used to find the likelihood of a given tree. A "wrapper" around this program has also been implemented to allow it to be used as a fitness function. This wrapper converts the tree representation explained in the last section to Newick format (see 2.4.1) and passes this tree along with the DNA sequences to PHYML. The output from PHYML is then parsed and the log-likelihood is returned.

The newest developments in PHYML are described in [13]. It is a highly optimized program and has a fast implementation of the maximum likelihood algorithm.

Chapter 5

Results

The system outlined in the previous chapter has been used on two data sets with mitochondrial genomes downloaded from the National Center for Biotechnology Information in the United States. The two data sets, both consisting of 20 mitochondrial genomes, are:

- `mamm20`: 20 mammalian, mitochondrial genomes
- `ray10mamm10`: 10 mammalian + 10 ray-finned fish mitochondrial genomes

Details about the genomes in each data set are listed in Appendix A on page 69.

Data sets with 20 genomes were chosen as this allowed testing on a personal workstation, while still representing a difficult optimization problem. As mentioned in Section 2.1.3 on page 9 there are 8.2×10^{21} possible rooted phylogenetic trees for 20 species.

The wrapper program around PHYML (described in Section 4.6 on the preceding page) was used as the fitness evaluator as this proved to be considerably faster (up to a factor of five) than the less optimized maximum likelihood function used in the testing phase.

Alignment was done using Clustal[1] (as explained in Appendix E on page 83) and gap sites were removed.

The tests were run on an cluster and used up to 7 nodes, each with 12 cores, making the total number of processor cores 84. The communication between the processes were facilitated by Simdist, as mentioned in Section 4.3 on page 42.

Two series of runs were performed on the first data set, the first one without differential reproduction (see Section 4.2.4 on page 41) and the second run allowing above-average parents to produce more offspring. The Kimura 80

[1] Clustal version 2.1.

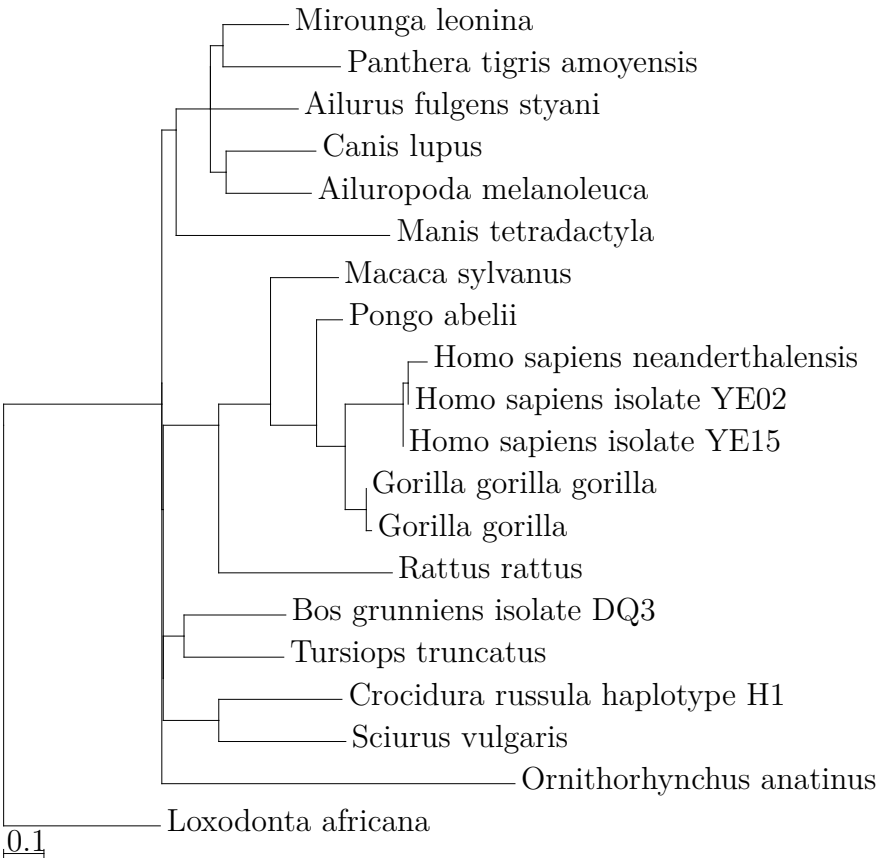


Fig. 5.1: The best tree found for the `mamm20` data set, with log likelihood of −152154.

model was used in the likelihood calculation and the transition/transversion ratio was set to 4.0. This number was estimated by PHYML. The mutation and recombination rates used in all runs were $p_c = 0.3^{\dagger}$ and $p_m = 0.7$. These values have not been systematically optimized, but were found to produce good results in a preliminary test of the system. The Prune-Delete-Graft operator was used for recombination. Mutation altered one random branch, adding a random value between -0.1 and 0.1 to its length and clamping the final value between 0 and 1.

Selection was done using rank-selection: each individual's probability of being selected to produce offspring was proportionate to its rank (0–20 in these data sets). Elitism was used in all runs, with the four best individuals getting copied to the next generation. Maximum likelihood will thus never decrease.

The running times for these tests are measured in CPU time. This is the sum of the time all processors spent running code, as opposed to waiting (e.g. for input/output operations). The run times is thus for an idealized scenario in which processors always work. In reality this was quite close to the real time it took. Fitness evaluation for one tree usually takes less than one second, so the system will never be waiting a long time for the likelihood score of a tree to be completed.

The best tree found for each set of runs by the EA system is shown. The branch lengths show the distance, measured in expected substitutions (see Section 2.1.5 on page 12).

The trees inferred by PHYML for the same data sets are included in Appendix B.

5.1 Twenty Mammalian Mitochondrial Genomes

The `mamm20` data set consists of the mitochondrial genomes of 20 mammals. After alignment and the removal of gap sites, the DNA sequences were 14659 base pairs. A phylogenetic tree inferred from these sequences by PHYML[3] is shown in Fig. B.1 on page 74 for illustration.

Inferring the tree with PHYML took 8 minutes and 9 seconds on a system with an AMD 2431 Istanbul processors running at 2400 MHz.

[†]This probability is for the whole genome, i.e. the probability of *one* random mutation happening to the genome as a whole.

[3]PHYML version `20110304`, command: `phyml -b 0 -i mamm20.phy -t 4.0 -m K80 -f "0.25,0.25,0.25,0.25" -t 4.0`

5.1.1 Runs

The EA system was run 20 times with the `mamm20` data set on the cluster described. The average and maximum fitness, respectively, per generation for each of the runs is shown in Fig. 5.2 on the following page. The runs were cut off after 300 generations. The number of generations to run for was set after it was observed that average fitness tended to level out a little before this in early runs with more generations.

The runs are quite consistent and produce trees with similar fitness ($\sigma_{\text{avg. fitness}} = 1560.14$, $\sigma_{\text{max fitness}} = 1498.39$), but as can be seen from Fig. 5.2b on the next page, the best tree found varies quite a bit between runs.

The log likelihood of the best tree found by the EA was -152154, while the tree inferred by PHYML had a log likelihood of -147188. The best tree found is shown in Fig. 5.1 on page 47. It has grouped the two *Homo sapiens* together, and the two closely related gorilla species together. All primates are correctly grouped together.

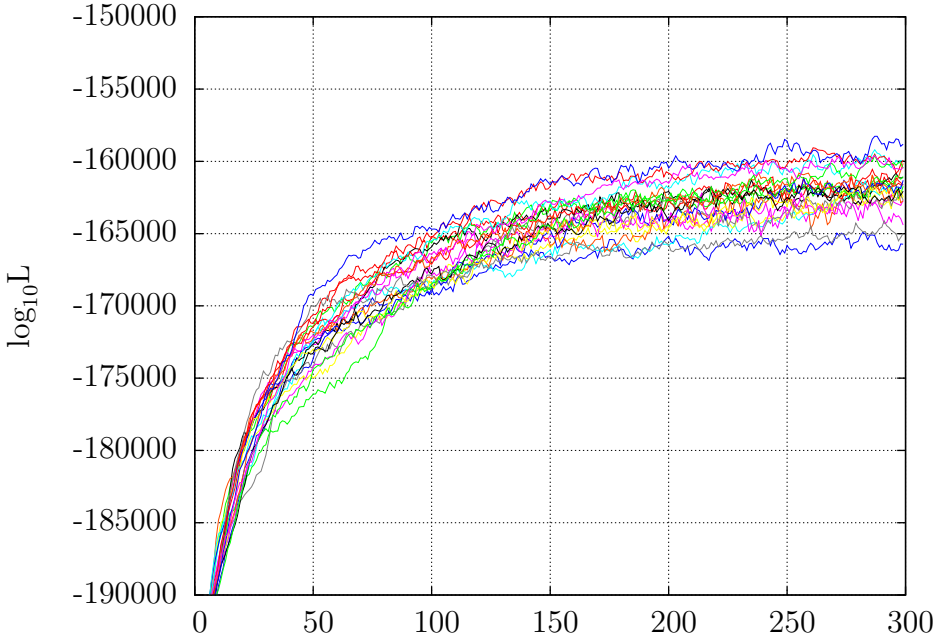
5.1.2 Dissecting a Run

To understand more about how the population evolves as a run progresses, a run that produced a tree with an average fitness among the 20 runs (the tenth best tree) was chosen and further studied. This run produced a tree with a log likelihood of -157497 after 300 generations. The maximum, average and minimum fitness per generation is shown in Section 5.1.1 . Some things are appearant from this plot:

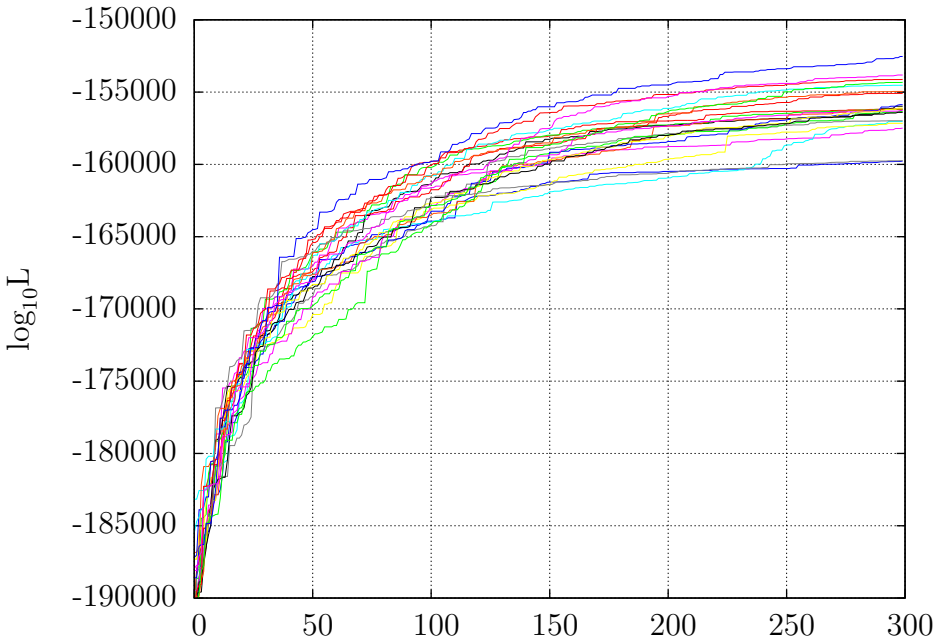
- The initial, random population improves rapidly during the first 50 or so generations as "fitter" trees are selected.
- There are not any big fluctuations in the average fitness after the initial generations.
- The minimum fitness can greatly vary from one generation to the next.

From the minimum fitness graph it is obvious that a recombination can have a large effect. Since a mutation is only a slight alteration of a branch length, the large fluctuations between the fitness of the lowest ranked individual in each generation must be caused by a recombination.

By looking at how many different topologies that are present in the population and the "success rate" of mutations and recombinations, we can understand more of how the artificial selection happens. Fig. 5.4 on page 52 shows the number of unique topologies as the run progressed. The average number of topologies is 228.7, indicated by the dashed line. Grouping by topologies



(a) Average fitness per generation.



(b) Maximum fitness per generation.

Fig. 5.2: Average and maximum fitness plots for 300 generations for the `mamm20` data set for 20 runs.

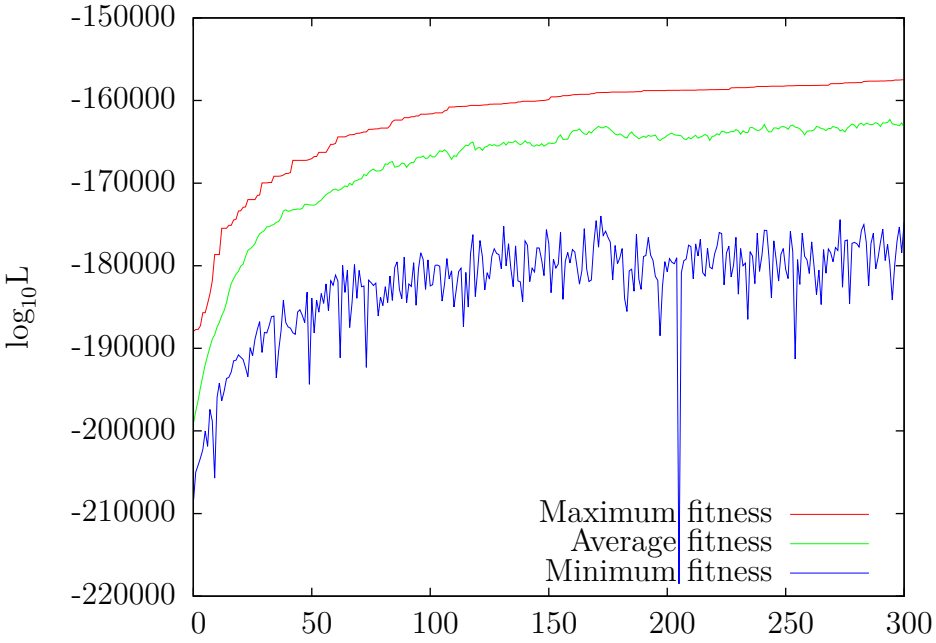


Fig. 5.3: Fitness plot for one of the runs on the `mamm20` data set.

was done by removing all labels and lengths from a tree's representation in the Newick format. This means that two trees that are in reality the same tree, such as $\curlywedge$ and $\curlywedge$ will be classified as two different topologies. As the plot shows, the number of different topologies seems to be relatively stable. This indicates that the population does not converge towards a limited set of local maxima due to the recombinations that happen.

The average success ratio of mutations and recombinations were also logged for this run. A "successful" mutation is defined as a mutation that makes the tree that is mutated have higher fitness after the mutations than before. In a population of random trees 50% of mutations should be expected to increase fitness, while in a population where the trees become more "fine-tuned", a random mutation should have a lower probability of being beneficial. This is exactly what we see in 5.5a. The ratio of mutations that lead to higher fitness starts at around 50% and drops as the run progresses. A "successful recombination" was chosen to be even more strictly defined: A recombination was chosen to be successful only if the offspring's fitness was better than *both* its parents. The number of such recombination is plotted in 5.5b. This number also drops rapidly as the population becomes more "fit".

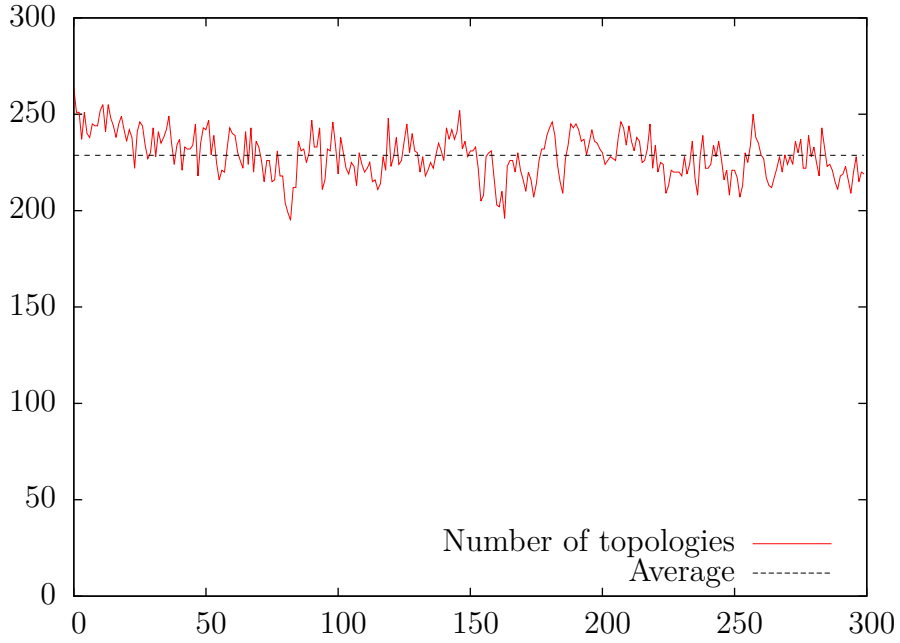


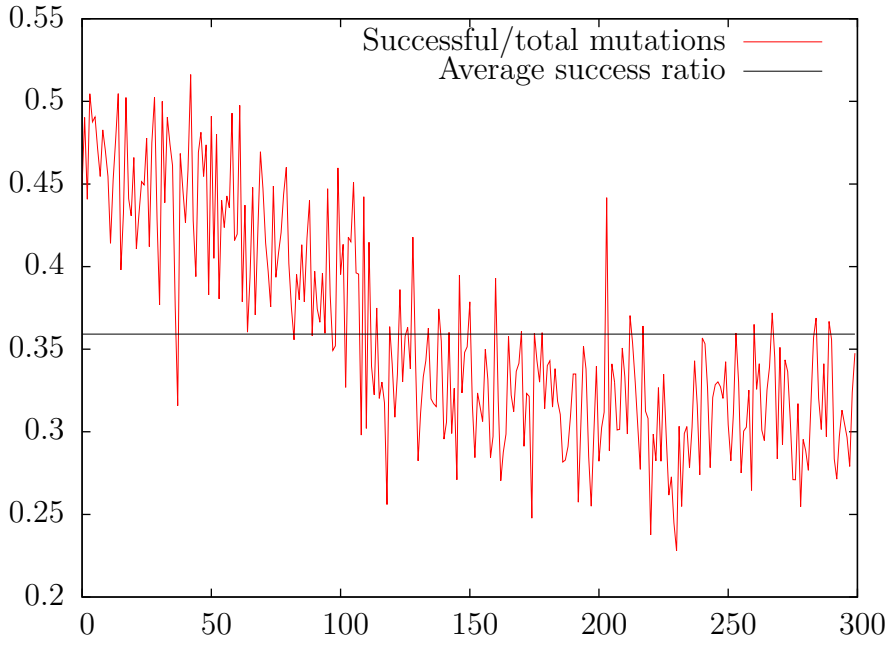
Fig. 5.4: The number of unique topologies of the population of 300 trees for each generation.

5.1.3 Differential Reproduction

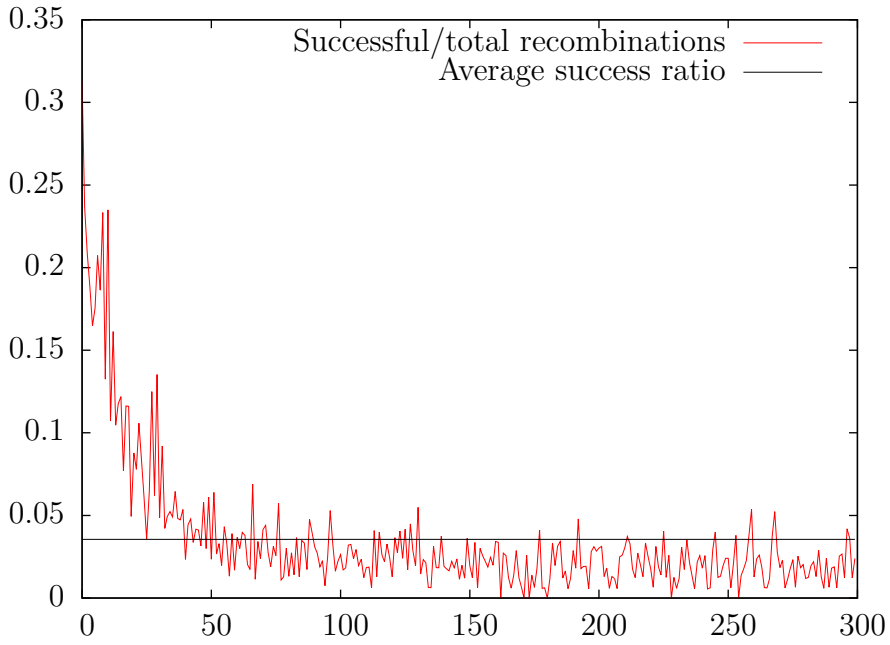
Inspired by Lewis in [23] who let the number of offspring produced be relative to parents' fitness, another 20 runs were done on the `mamm20` data set, this time letting parents produce one extra offspring for each parent that was above average. Two above-average parents will thus pass on 3 offspring to the next generation. The average and maximum fitness for these 20 runs are shown in Fig. 5.6 on page 54.

The results are very similar to the results obtained without using differential reproduction. If allowing above-average parents to have more offspring had an effect, it was a small one. The number of unique topologies were not affected either. The average number of unique toplogies was 200.6, a little lower than in the first set of runs, but this does not appear to have a significant effect on the best tree found.

The best tree found in these runs is shown in Fig. 5.8 on page 56 and has a fitness of -152018.

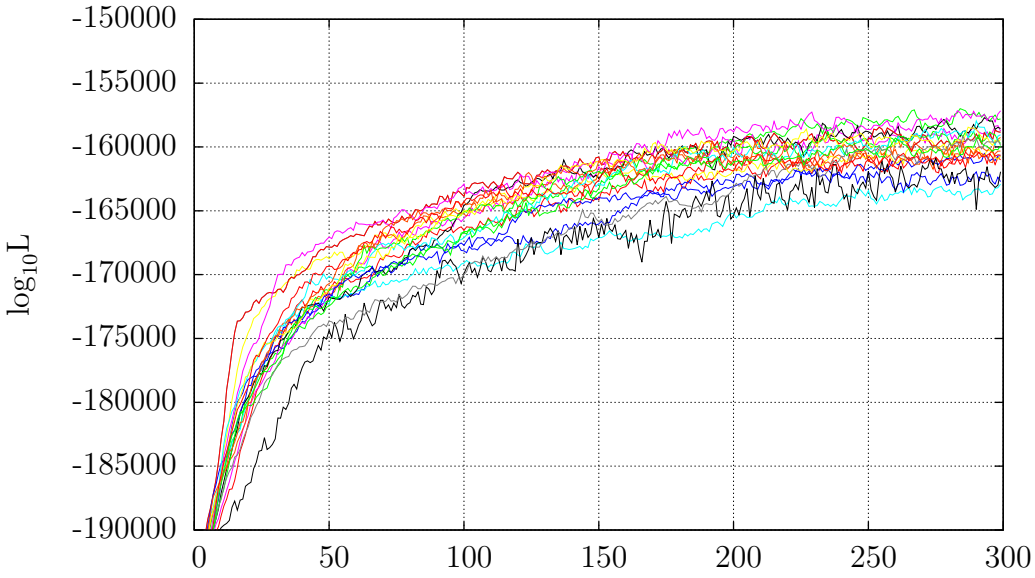


(a) Successful mutations

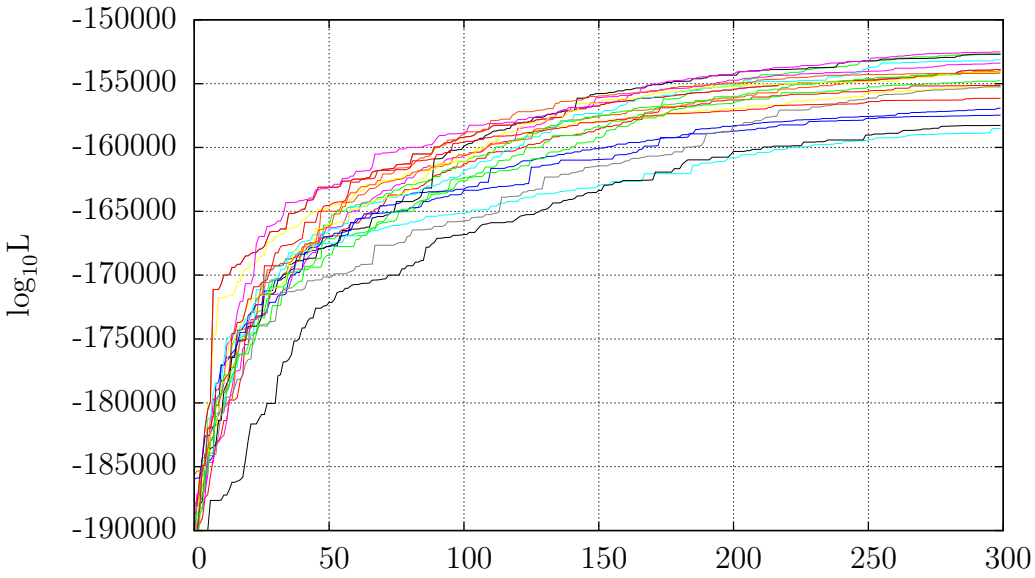


(b) Successful recombinations

Fig. 5.5: "Success rates" for mutations (top) and recombinations (bottom) as the run progresses.



(a) Average fitness per generation.



(b) Maximum fitness per generation.

Fig. 5.6: Average and maximum fitness plots for 300 generations for the `mamm20` data set for 20 runs with differential reproduction.

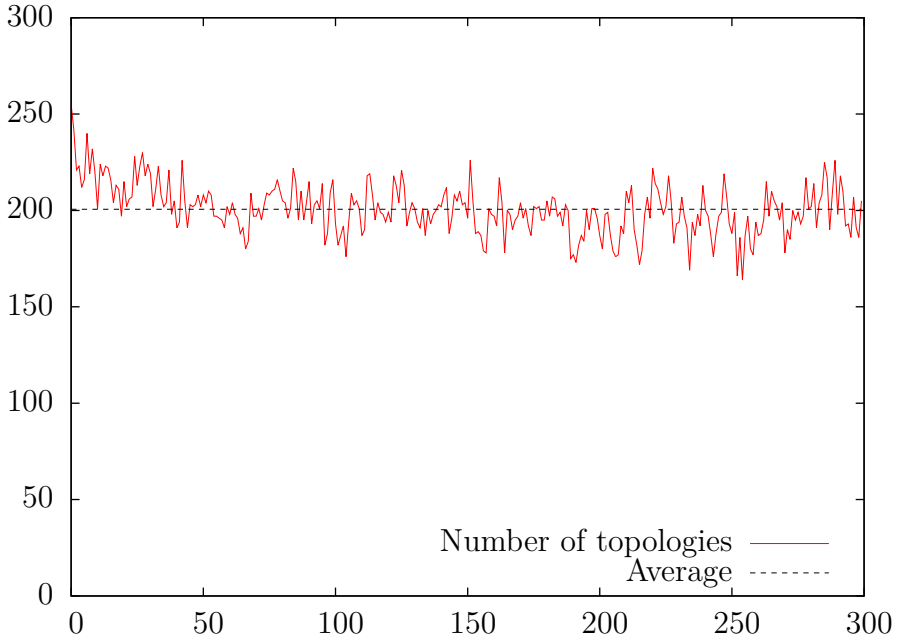


Fig. 5.7: The number of unique topologies of the population of 300 trees for each generation when using differential reproduction.

5.2 Ten Ray-Finned Fishes, Ten Mammals

The `ray10mamm10` data set consists of the mitochondrial genomes of 10 mammals and 10 ray-finned fishes. After alignment and the removal of gap sites, the DNA sequences were 15006 base pairs. A phylogenetic tree inferred from these sequences by PHYML[4] is shown in Fig. B.2 on page 75 for illustration. The tree inferred by PHYML has a log likelihood of −152252 and has correctly grouped the fishes close together and separate from the mammals. Within the mammals, the primates are grouped closely together.

Inferring the tree with PHYML took 16 minutes and 38 seconds on a system with an AMD 2431 Istanbul processors running at 2400 MHz.

The EA system was run 20 times with the `ray10mamm10` data set on the cluster. The same differential reproduction scheme used in the second `mamm20` run was used. The average and maximum fitness, respectively, per generation for each of the runs is shown in Fig. 5.10 on page 59. The runs were again cut off after 300 generations.

The runs are also consistent for this data set and produce trees with

[4]PHYML version `20110304`, command: `phyml -b 0 -i ray10mamm10.phy -t 4.0 -m K80 -f "0.25,0.25,0.25,0.25" -t 4.0`

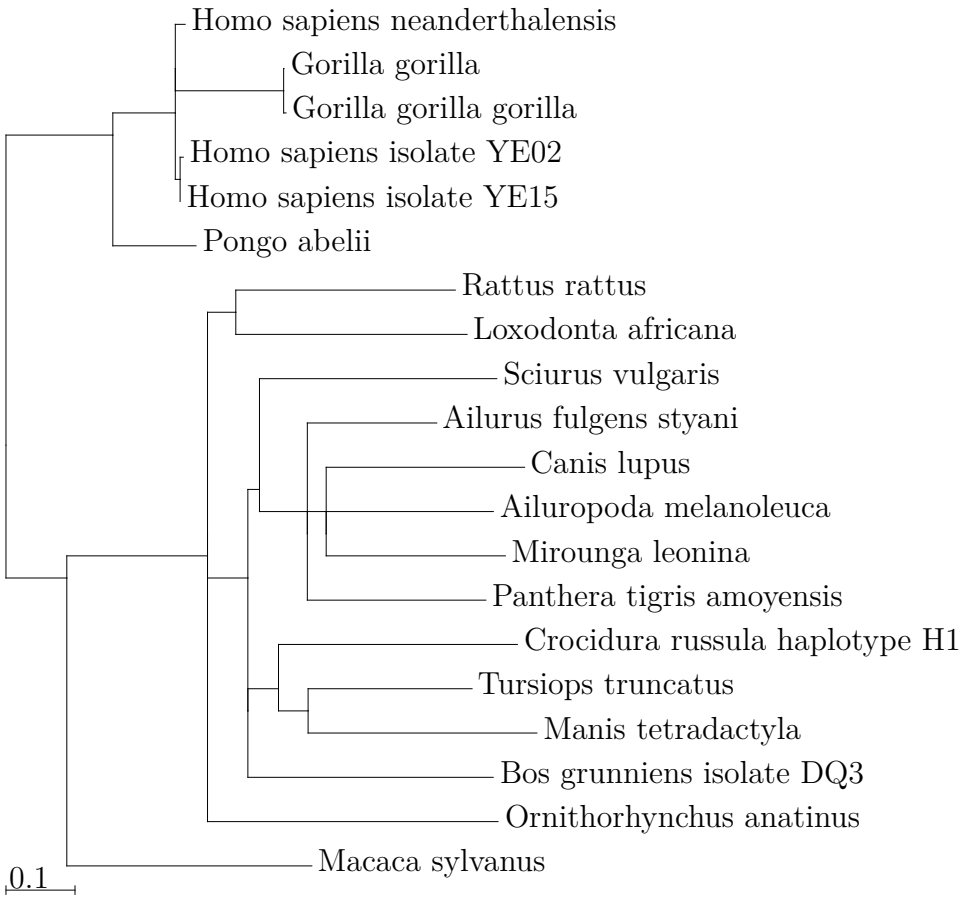


Fig. 5.8: The best tree found for the `mamm20` data set with differential reproduction, with a log likelihood of−152018.

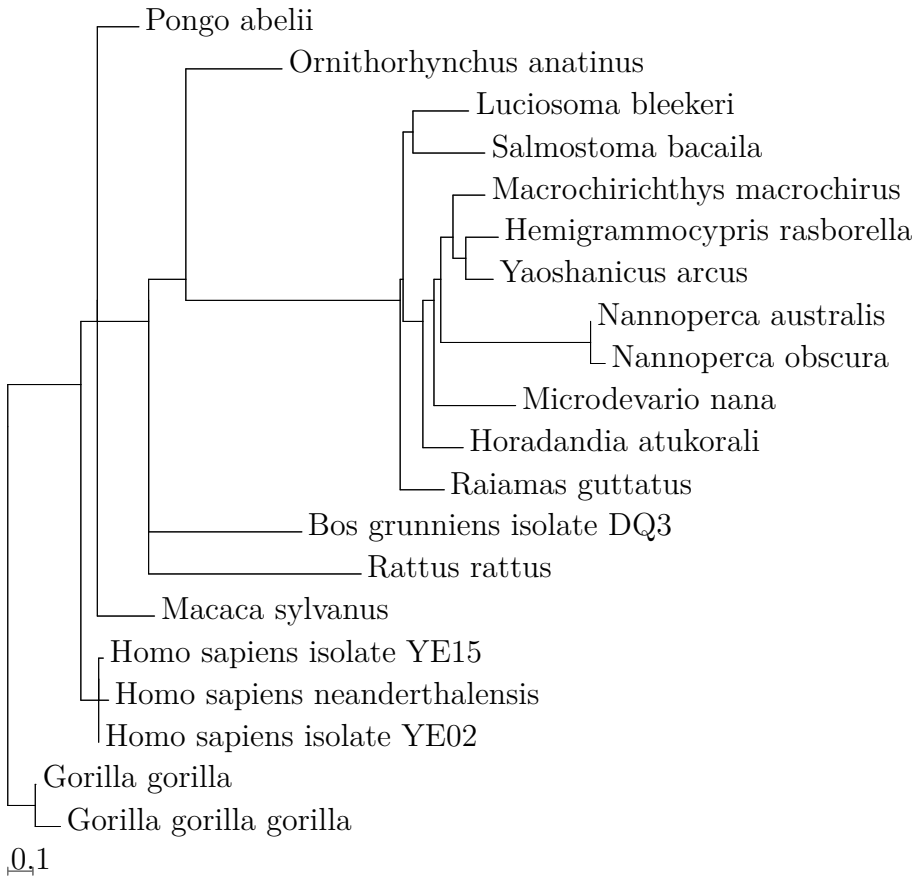


Fig. 5.9: The best tree found for the `ray10mamm10` data set with differential reproduction, with a log likelihood of−158556.

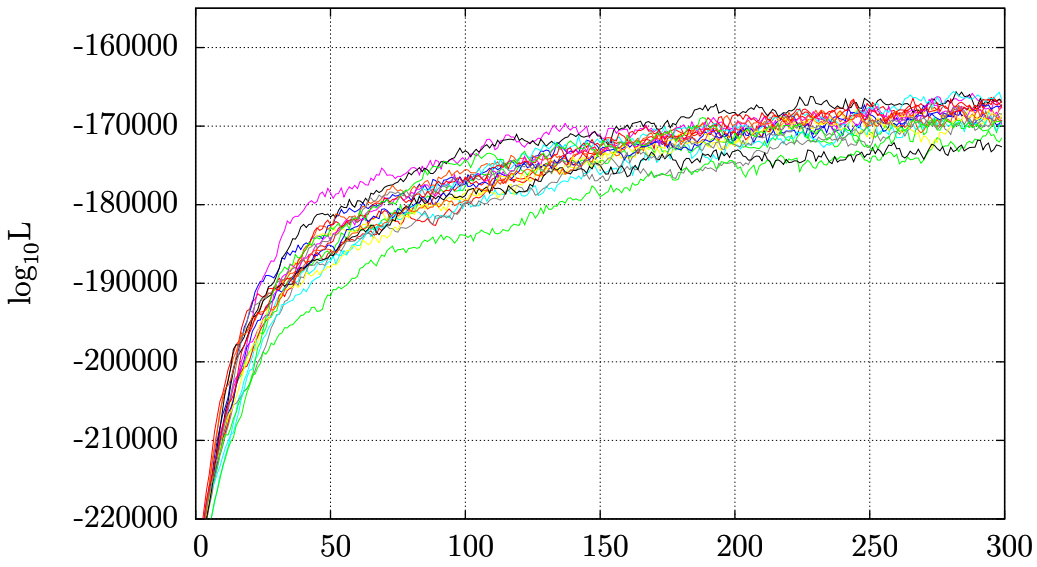
similar fitness ($\sigma_{\text{avg. fitness}} = 1587.65$, $\sigma_{\text{max fitness}} = 1564.27$).

The log likelihood of the best tree found by the EA was −158556. The tree is shown in Fig. 5.9 on the previous page. The tree inferred by PHYML had a log likelihood of −152252. The best tree found is shown in Fig. 5.1 on page 47. It has correctly partioned the tree into ray-finned fishes and mammals.

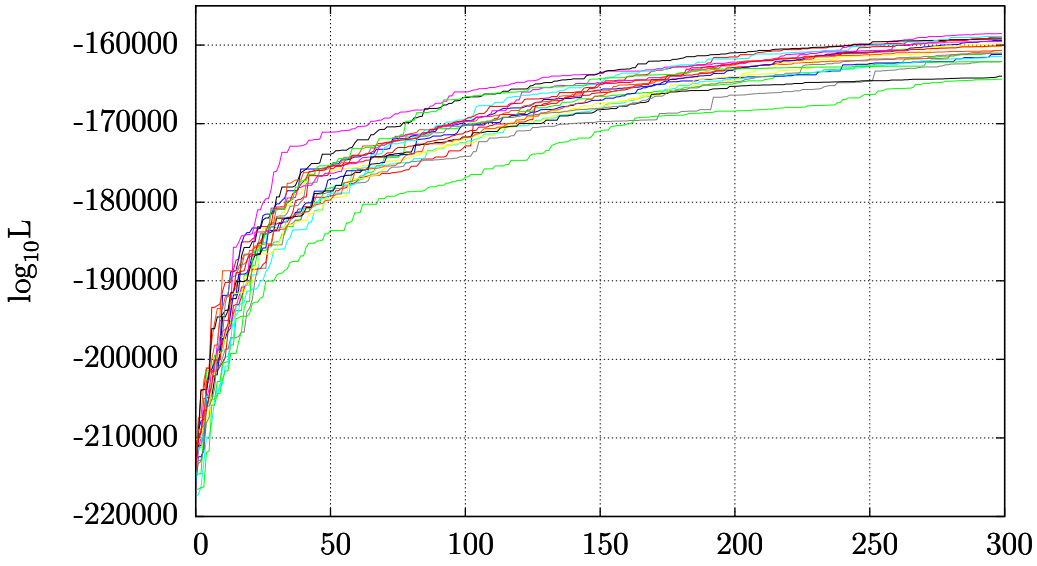
5.3 Running Time

The average CPU time used run the EA for 300 generations on the `mamm20` data set was 74 minutes and 27 seconds. Using differential reproduction did not seem to affect this, and the 20 runs using this had an average CPU time usage of 76 minutes 22 seconds. The average CPU time used per run on the `ray10mamm10` data set was 105 minutes and 35 seconds.

These running times are considerably higher than PHYML's running time on the same data sets, so the search needs to be greatly improved in order to make an EA system an attractive alternative for data sets of this size. Some suggestions for further work that can be done is given in the last section of the conclusions chapter.



(a) Average fitness per generation.



(b) Maximum fitness per generation.

Fig. 5.10: Average and maximum fitness plots for 300 generations for the `ray10mamm10` data set for 20 runs. The fitness values are the logarithm of the individuals' likelihood scores.

Chapter 6

Conclusion and Further Work

The amount of available molecular data (e.g. in form of DNA sequences) has grown rapidly the last couple of decades as sequencing technology has improved. Part of the credit for this is due to the Human Genome Project which in May 2006 completed the sequence of the last chromosome of the Human genome, after having announced a "rough draft" genome in 2000.

As the amount of molecular data accrues, efficient computers and algorithms become increasingly important to analyze this data. The problem of inferring phylogenies directly from molecular data goes to the core of this problem. Using a maximum-likelihood with models for nucleotide evolution allows us to use all the sequence data. This is a computationally expensive approach, but the rapid improvements in computer technology has made it feasible.

The number of possible trees for a set of species grows so fast that it is necessary to use heuristic search methods to find a good hypotheses for the true tree. Evolutionary algorithms is a class of such search/optimization algorithms that has been shown to perform well in other areas where the search space is large and irregular. A firm conclusion about the suitability of EAs for this area cannot be drawn at this point.

> *The main results of this thesis is that a relatively simple EA system is able to come up with good hypotheses for phylogenies by only using DNA sequence data, and that this system makes it very easy to take advantage of a parallel computer system.*

A Bigger Needle in a Bigger Haystack

Great care has been taken in making sure that the likelihood evaluation of trees has a solid base in biology/genetics theory. There are still some sources of error, but few that are unique to this system.

There are many inherently difficult problems with phylogenetic inference, and these problems are not necessarily improved with having more sequence data. The substitutions occurring along a branch of a phylogenetic tree is often referred to as the *phylogenetic signal*. In branches with many changes the signal is proportionally strong. Noise that interferes with the genuine phylogenetic signal is called *non-phylogenetic signal* and "competes" with phylogenetic signal during phylogenetic inference. Even when using considerable amounts of sequence data, geneticists still get highly incongruent results when inferring phylogenies [29]. More sequence data is not enough to solve these phylogenetic problems due to several factors. One factor is speciation events[1] that are close in time, leading to short internal branches in the tree that are difficult to resolve. Another problem is that when we have events of interest that are ancient, the branches down to leaf nodes are often very long and have had multiple substitutions happen at the same site. This is very difficult to detect.

EA Parameters

From the EA side, much can be done to further improve results. Finding the best parameters, selection function, and mutation and recombination operators can be a science of its own. It is probably possible to improve performance by tuning parameter values and selection strategies. It could even be the case that different data sets need different parameters. Both data sets that were used for testing were of species that are quite closely related when looking at the whole spectre of life forms. Data sets where the distance between species are large may require different parameters and operators.

It is hard to decide which pathways to follow in such a wide field. Some suggestions are offered in the following section on further work.

Parallelization

Parallelizing the system was easy once some initial synchronization problems were solved. By using Simdist to distribute the computations, all available processors – up to the population size of the EA – will be utilized. In an ideal scenario where the number of available processors are equal to the population size, fitness evaluation of each generation would only take a time equal to the maximum time of evaluating one individual. For the data sets of 20 sequences used here, that number was around 0.5 seconds.

[1] Two or more lineages separating into what will become different species.

6.1 Further Work

EA "Tuning"

During an EA run the main concern is making sure that the individuals are freely exploring the search space. Mutations and recombinations create a diverse population, and individuals in promising areas of the search space have a higher probability to contribute offspring – with modifications – to the next generations. These offspring have a relatively high probability of inhering good properties from their parents and thus further explore around these areas. As the run progresses, the population will tend to converge around these local maxima. In this phase, recombinations and mutation that only slightly "move" the individuals are desireable since we want to find the "highest" point of the maxima.

A common way to accomplish this is to reduce the probability of recombinations and mutations at the end of a run. This is often done by introducing a "temperature" that starts high and decreases as the run progresses. This temperature is used to control the probability of mutations and recombinations. In the case of evolution of phylogenetic trees with mutations affecting branch lenghts, it could perhaps also be good to reduce the *magnitude* of mutations towards the end of a run.

Statistical Confidence

Because of the large number of possible tree topologies, it would be beneficial to assess how much confidence one can have that a particular tree is a good estimation of a phylogeny. A widely used method of assessing confidence in the nodes of a tree is the *bootstrap test* [9]. The basic idea behind the bootstrap test is to repeatedly draw a subset of the original data, build a tree based on this data, and then check how many of the subtrees that overlap. Another, newer method is the *approximate likelihood ratio test*, or aLRT introduced in 2006 [1] which is much cheaper to compute in maximum likelihood systems since it reuses the results of computations already performed during the ML calculations. This method has been shown to estimate the probability of a branch being correct better than the bootstrapping method [1, 14].

Having a number denoting the statistical confidence we can place in a tree would be of help both during the "tuning" described above and when using the system. We want to infer phylogenies from data where the true phylogeny is unknown, so there is no answer we can compare it to. A statistics test could indicate to which degree the "hypothesis trees" that are produced should be trusted.

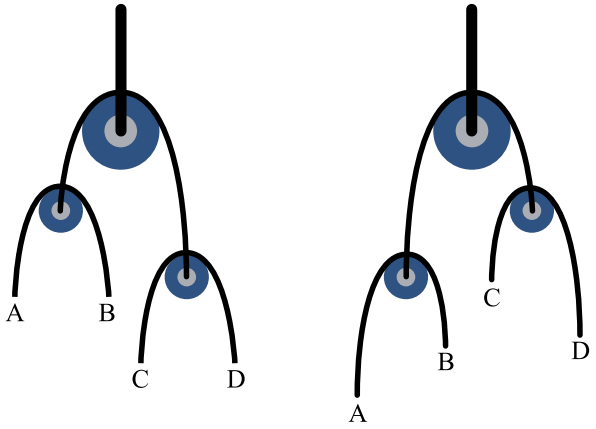


Fig. 6.1: The pulley principle says that the root can be placed anywhere along a branch without affecting the tree's likelihood if the model of evolution is reversible. The tree in (a) will thus have the same likelihood as the tree in (b).

More Advanced Models of Nucleotide Evolution

The Kimura 80 model that was used as the model of nucleotide evolution in the maximum-likelihood estimation is quite simple. Other models such as the HKY and F84 models extend the Kimura two-parameter model to allow different prior probabilities of the four different bases. Even more complex models exist and it could also be interesting to allow the EA to modify the model parameters too.

Optimizing Branch Lenghts as part of Fitness Evaluation

If the evolutionary model $P(i \mid j, t)$ is reversible, i.e. $P(i \mid j, t) = P(j \mid i, t)$, it can be shown that the placement of the root does not affect a tree's likelihood. This is called the *Pulley principle* and is illustrated in Fig. 6.1 . Using this principle makes it possible to optimize the branch lengths of a topology by placing the root along branches, in turn, and then moving the root in an iterative fashion, towards what results in the highest likelihood [8]. The Expectation maximization algorithm can be used to carry out this iterative search [10].

By optimizing the branch length of a tree topology, branch lenghts need not be evolved and the search space will be that of all possible tree topologies.

PHYML supports optimizing branch lenghts and since the Kimura 80 model is reversible some initial tests were performed by modifying the EA

system to evolve trees without branch lenghts by using PHYML to optimize branches and return the maximum log likelihood for a tree topology. This was done by changing the mutation operator to swap two random leaf nodes, while the Prune-Graft-Delete recombination operator was kept. However, evaluating a tree took 150–250 seconds, which would result in runs with a large population of trees taking far too many resources.

This can perhaps work better if (1) it were possible to detect poor trees early in the fitness evaluation and stop the meticulously calculation of an exact likelihood for trees that are not promising, or (2) the optimization could be made faster by only iterating over a branch a limited amount of times. The latter option could even be modified throughout the run, spending more time on finding the truly optimal branch lenghts late in the run, while only approximating the "scores" for trees in the early generations.

Bibliography

[1] Anisimova, M., and Gascuel, O. Approximate likelihood-ratio test for branches: A fast, accurate, and powerful alternative. *Systematic biology 55*, 4 (Aug. 2006), 539–52.

[2] Asanovic, K., Bodik, R., Catanzaro, B., Gebis, J., Husbands, P., Keutzer, K., Patterson, D., Plishker, W., Shalf, J., Williams, S., and Others. The landscape of parallel computing research: A view from berkeley. Tech. rep., Citeseer, 2006.

[3] Bryant, D., Galtier, N., and Poursat, M.-A. Likelihood Calculation in Molecular Phylogenetics. In *Mathematics of Evolution and Phylogeny*. Oxford University Press, 2005, ch. 2.

[4] Bungum, L., and Gambäck, B. Evolutionary Algorithms in Natural Language Processing. In *Proceedings of the second Norwegian Artificial Intelligence Symposium* (2010), no. November, pp. 7–19.

[5] Chenna, R., Sugawara, H., Koike, T., Lopez, R., Gibson, T., Higgins, D., and Thompson, J. Multiple sequence alignment with the Clustal series of programs. *Nucleic acids research 31*, 13 (July 2003), 3497–3500.

[6] Cotta, C., and Moscato, P. Inferring phylogenetic trees using evolutionary algorithms. *Parallel Problem Solving from Nature–PPSN VII* (2002), 720–729.

[7] Downing, K. L. Introduction to Evolutionary Algorithms, 2009.

[8] Felsenstein, J. Evolutionary trees from DNA sequences: a maximum likelihood approach. *Journal of molecular evolution 17*, 6 (Jan. 1981), 368–76.

[9] Felsenstein, J. Confidence limits on phylogenies: An approach using the bootstrap. *Evolution 39*, 4 (1985), 783–791.

[10] Felsenstein, J. *Inferring phylogenies*. Sinauer Associates, Inc. Publishers Sunderland, Massachusetts, 2004. Fourth printing.

[11] Floreano, D., and Mattiussi, C. *Bio-Inspired Artificial Intelligence*. MIT Press, 2008.

[12] Gottlieb, J., Julstrom, B. A., Raidl, G. R., and Rothlauf, F. Prüfer Numbers: A Poor Representation of Spanning Trees for Evolutionary Search. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)* (2001), Citeseer, pp. 343–350.

[13] Guindon, S., Dufayard, J. F., Lefort, V., Anisimova, M., Hordijk, W., and Gascuel, O. New Algorithms and Methods to Estimate Maximum-Likelihood Phylogenies: Assessing the Performance of PhyML 3.0. *Systematic Biology 59*, 3 (Mar. 2010), 307–321.

[14] Hall, B. *Phylogenetic trees made easy: a how-to manual*. Sinauer Associates, Inc. Publishers Sunderland, Massachusetts, 2004.

[15] Han, M. V., and Zmasek, C. M. phyloXML: XML for evolutionary biology and comparative genomics. *BMC bioinformatics 10* (Jan. 2009), 356.

[16] Hartl, D., and Clark, A. *Principles of population genetics*. Sinauer Associates, Inc. Publishers Sunderland, Massachusetts, 1997.

[17] Hassan, R., Hossain, M. M., and Karmakar, C. K. Phylogeny Inference Using a Multi-objective Evolutionary Algorithm with Indirect. 41–50.

[18] Hø verstad, B. A. Simdist: a distribution system for easy parallelization of evolutionary computation. *Genetic Programming and Evolvable Machines 11*, 2 (Jan. 2010), 185–203.

[19] Huelsenbeck, J., Bollback, J., and Levine, A. Inferring the root of a phylogenetic tree. *Systematic biology 51*, 1 (2002), 32–43.

[20] Huelsenbeck, J. P., and Crandall, K. A. Phylogeny Estimation and Hypothesis Testing Using Maximum Likelihood. *Annual Review of Ecology and Systematics 28* (1997), 437–466.

[21] Hugall, A. F., and Lee, M. S. Y. The likelihood node density effect and consequences for evolutionary studies of molecular rates. *Evolution; international journal of organic evolution 61*, 10 (Oct. 2007), 2293–307.

[22] Junier, T., and Zdobnov, E. M. The Newick utilities: high-throughput phylogenetic tree processing in the UNIX shell. *Bioinformatics 26*, 13 (July 2010), 1669–70.

[23] Lewis, P. O. A genetic algorithm for maximum-likelihood phylogeny inference using nucleotide sequence data. *Molecular biology and evolution 15*, 3 (Mar. 1998), 277–83.

[24] Marlow, S. *Parallel and Concurrent Programming in Haskell*. Microsoft Research Ltd., Cambridge, U.K., 2011.

[25] Matsuda, H. Protein Phylogenetic Inference Using Maximum Likelihood with a Genetic Algorithm. *Pacific Symposium on Biocomputing* (1996), 512–523.

[26] Mitchell, M. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[27] Perelman, P., Johnson, W. E., Roos, C., Seuánez, H. N., Horvath, J. E., Moreira, M. a. M., Kessing, B., Pontius, J., Roelke, M., Rumpler, Y., Schneider, M. P. C., Silva, A., O'Brien, S. J., and Pecon-Slattery, J. A Molecular Phylogeny of Living Primates. *PLoS Genetics 7*, 3 (Mar. 2011), e1001342.

[28] Pettersson, E., Lundeberg, J., and Ahmadian, A. Generations of sequencing technologies. *Genomics 93*, 2 (Feb. 2009), 105–11.

[29] Philippe, H., Brinkmann, H., Lavrov, D. V., Littlewood, D. T. J., Manuel, M., Wörheide, G., and Baurain, D. Resolving Difficult Phylogenetic Questions: Why More Sequences Are Not Enough. *PLoS Biology 9*, 3 (Mar. 2011), e1000602.

[30] Rogers, J. S., and Swofford, D. L. Multiple local maxima for likelihoods of phylogenetic trees: a simulation study. *Molecular biology and evolution 16*, 8 (Aug. 1999), 1079–85.

[31] Rothlauf, F., and Goldberg, D. E. Pruefer Numbers and Genetic Algorithms : A Lesson on How the Low Locality of an Encoding Can Harm the Performance of GAs. *Lecture Notes in Computer Science 1917* (2000), 395–404.

[32] Rubin, E., Lucas, S., Richardson, P., Rokhsar, D., and Pennacchio, L. Finishing the euchromatic sequence of the human genome. *Nature 431*, LBNL–57963 (Feb. 2004).

[33] Russell, P. *iGenetics: a molecular approach*. Pearson/Benjamin Cummings, 2006.

[34] Russell, S., and Norvig, P. *Artificial Intelligence: A Modern Approach*, 2nd ed. ed. Prentice Hall, May 2003.

[35] Shendure, J., and Ji, H. Next-generation DNA sequencing. *Nature biotechnology 26*, 10 (Oct. 2008), 1135–45.

[36] Steel, M. A. The maximum likelihood point for a phylogenetic tree is not unique. *Systematic Biology 43*, 4 (1994), 560–564.

[37] Theobald, D. L. A formal test of the theory of universal common ancestry. *Nature 465*, 7295 (May 2010), 219–222.

[38] Venditti, C., Meade, A., and Pagel, M. Detecting the node-density artifact in phylogeny reconstruction. *Systematic biology 55*, 4 (Aug. 2006), 637–43.

Appendix A

Data sets

A.1 `mamm20`

The `mamm20` data set consists of 20 mammalian mitochondrial genomes, available from the National Center for Biotechnology Information in the US. They are listed below in no particular order.

1. *Pongo abelii* (Sumatran orangutan)

   Pongo abelii mitochondrion, complete genome:

   http://www.ncbi.nlm.nih.gov/nuccore/NC_002083

2. *Gorilla gorilla gorilla (*Western lowland gorilla)

   Gorilla gorilla gorilla mitochondrion, complete genome:

   http://www.ncbi.nlm.nih.gov/nuccore/NC_011120.1

3. *Gorilla gorilla* (Western gorilla)

   Gorilla gorilla mitochondrion, complete genome:

   http://www.ncbi.nlm.nih.gov/nuccore/NC_001645.1

4. *Homo sapiens neanderthalensis* (Neanderthal)

   Homo sapiens neanderthalensis complete mitochondrial genome, isolate Mezmaiskaya 1:

   http://www.ncbi.nlm.nih.gov/nuccore/FM865411.1

5. *Ornithorhynchus anatinus (*Platypus)

   Ornithorhynchus anatinus mitochondrion, complete genome

   http://www.ncbi.nlm.nih.gov/nuccore/5836058

6. *Bos grunniens (*Yak)
   Bos grunniens isolate DQ3 mitochondrion, complete genome
   http://www.ncbi.nlm.nih.gov/nuccore/GQ464314.1

7. *Rattus rattus* (Black rat)
   Rattus rattus mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/FJ355927.1

8. *Homo sapiens* [I] (Human)
   Homo sapiens isolate YE02 mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/318039821

9. *Homo sapiens* [II] (Human)
   Homo sapiens isolate YE15 mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/318039835

10. *Macaca sylvanus* (Barbary macaque)
    Macaca sylvanus mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/NC_002764.1

11. *Canis lupus (*Grey wolf)
    Canis lupus mitochondrial DNA, complete genome, haplotype: Jw258
    http://www.ncbi.nlm.nih.gov/nuccore/AB499825.1

12. *Panthera tigris amoyensis (*South China tiger)
    Panthera tigris amoyensis mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/NC_014770.1

13. *Ailuropoda melanoleuca (*Giant panda)
    Ailuropoda melanoleuca mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/EF196663.1

14. *Ailurus fulgens styani (*Styan's Red panda)
    Ailurus fulgens styani mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/NC_009691.1

15. *Tursiops truncatus* (Bottlenosed dolphin)
    Tursiops truncatus mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/NC_012059.1

16. *Sciurus vulgaris (*Eurasian red squirrel)
    Sciurus vulgaris mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/NC_002369.1

17. *Crocidura russula* (White-toothed shrew)
    Crocidura russula haplotype H1 mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/AY769263.1

18. *Manis tetradactyla* (Long-tailed pangolin)
    Manis tetradactyla mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/NC_004027.1

19. *Loxodonta africana* (African savannah elephant)
    Loxodonta africana mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/NC_000934.1

20. *Mirounga leonina* (Southern elephant seal)
    Mirounga leonina mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/115494593

A.2 euteleostomi20

The `euteleostomi20` data set contains the ten mammalian mitochondrial genomes listed as number 1–10 in the `mamm20` data set in addition to the following list of ten mitochondrial genomes.

1. *Macrochirichthys macrochirus* (Sward minnow)
   Macrochirichthys macrochirus mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/NC_015551.1

2. *Salmostoma bacaila* (Large razorbelly minnow)
   Salmostoma bacaila mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/NC_015549.1

3. *Hemigrammocypris rasborella**
   Hemigrammocypris rasborella mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/NC_015548.1
4. *Raiamas guttatus* (Burmese trout)
   Raiamas guttatus mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/NC_015547.1
5. *Microdevario nana**
   Microdevario nana mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/NC_015546.1
6. *Nannoperca obscura**
   Nannoperca obscura mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/NC_015545.1
7. *Horadandia atukorali**
   Horadandia atukorali mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/NC_015544.1
8. *Nannoperca australis**
   Nannoperca australis mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/NC_015542.1
9. *Luciosoma bleekeri**
   Luciosoma bleekeri mitochondrion, complete genome:
   http://www.ncbi.nlm.nih.gov/nuccore/NC_015541.1
10. *Yaoshanicus arcus mitochondrion**
    Yaoshanicus arcus mitochondrion, complete genome:
    http://www.ncbi.nlm.nih.gov/nuccore/NC_015540.1

*No common English name is registered for this species.

Appendix B

PHYML Inferred Trees

The following trees were inferred by PHYML for the `mamm20` and `ray10mamm10` data sets, respectively.

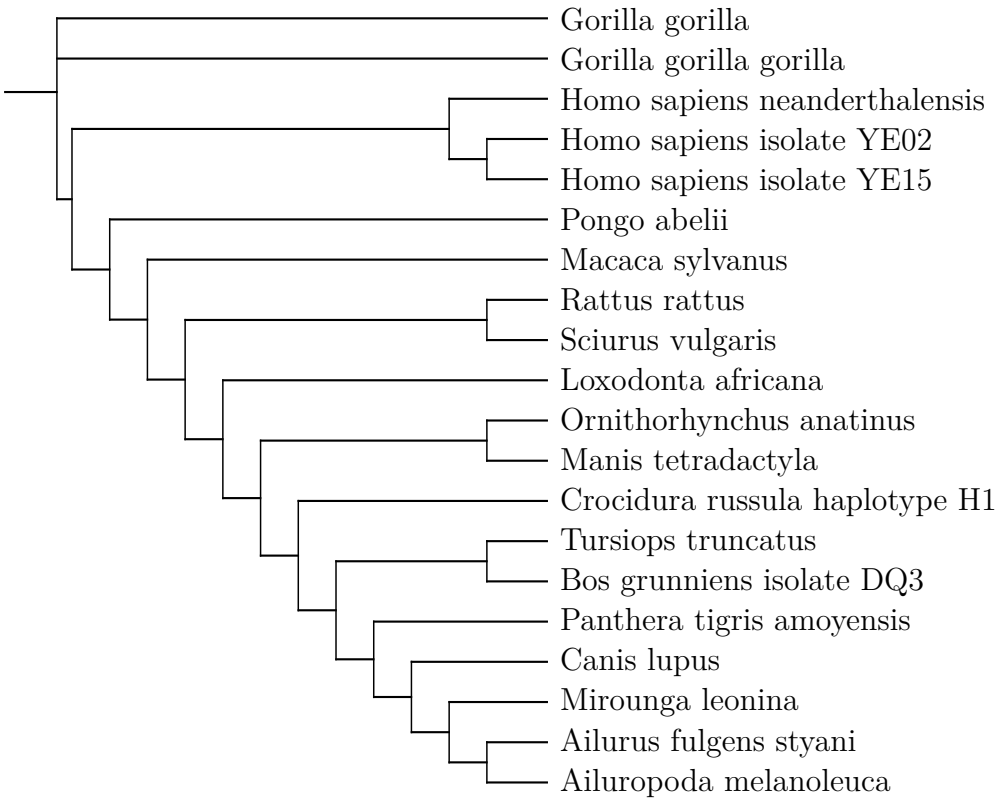


Fig. B.1: The phylogenetic tree inferred by PHYML for the `mamm20` data set.

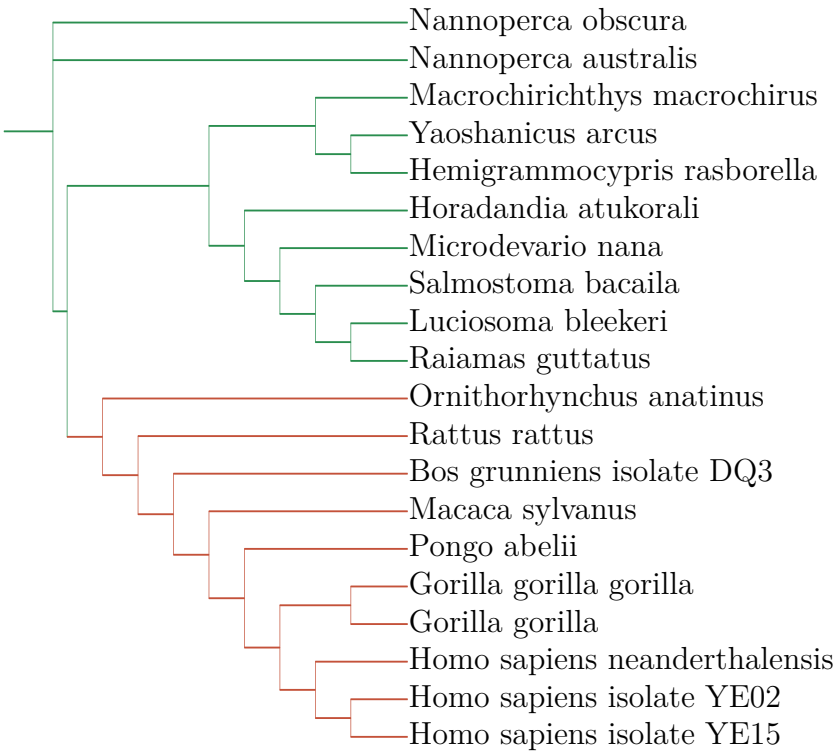


Fig. B.2: The phylogenetic tree inferred by PHYML for the `ray10mamm10` data set. The group of ray-finned fishes has been coloured green, while mammals are red.

Appendix C

Fitness Values

The following pages list the average, maximum and minimum $\log_{10}$-likelihoods for the last generation for the three sets of runs shown in Chapter 5:

1. EA version 1 on the `mamm20` data set
2. EA version 2 on the `mamm20` data set
3. EA version 3 on the `ray10mamm10` data set

	Average	Maximum	Minimum
1	-161350	-156183	-174524
2	-162611	-156196	-213045
3	-166143	-159796	-182890
4	-162970	-157523	-176998
5	-162020	-157043	-177415
6	-162678	-157161	-174075
7	-161121	-155983	-173249
8	-162077	-156185	-186679
9	-162340	-156983	-181473
10	-161511	-155097	-192296
11	-161101	-156345	-172957
12	-161781	-155896	-178586
13	-163768	-156317	-186936
14	-160109	-154535	-185430
15	-161913	-156104	-189934
16	-161886	-156406	-176942
17	-161099	-154957	-174873
18	-165501	-159757	-185794
19	-160646	-154130	-184469
20	-160092	-154339	-173141

Table C.1: EA v1, `mamm20`

	Average	Maximum	Minimum
1	-161015	-155144	-182462
2	-157804	-152681	-211089
3	-160701	-156923	-171540
4	-158761	-153394	-182222
5	-162913	-158515	-176616
6	-160712	-155280	-173048
7	-158554	-152694	-183692
8	-159253	-153944	-179816
9	-158849	-153901	-174038
10	-158849	-153901	-174038
11	-159731	-154032	-188156
12	-162718	-157468	-175381
13	-157163	-152517	-171274
14	-159049	-153129	-172768
15	-159409	-154077	-178240
16	-161936	-158269	-171243
17	-160320	-154154	-187092
18	-159861	-155239	-172617
19	-160642	-156148	-174728
20	-159898	-154769	-173979

Table C.2: EA v2, `mamm20`

	Average	Maximum	Minimum
1	-166718	-159107	-191047
2	-170002	-161091	-193802
3	-169233	-161201	-196640
4	-167614	-158556	-191892
5	-169399	-161452	-206188
6	-169069	-160203	-195727
7	-166996	-159234	-186557
8	-167288	-160673	-196288
9	-168996	-162077	-190040
10	-166825	-158945	-196692
11	-171501	-164347	-192351
12	-167279	-159406	-186601
13	-169824	-159572	-196182
14	-166728	-158902	-189893
15	-168371	-159840	-193627
16	-172651	-163955	-200266
17	-169322	-160725	-199120
18	-168494	-161008	-193895
19	-167655	-160112	-193645
20	-169493	-162115	-197917

Table C.3: EA v2, `ray10mamm10`

Appendix D

Distributed computations on a cluster

I have used the *Simdist* software described in [18] to run simulations on a cluster – a group of linked computers – in order to speed up evaluation of artificial genomes in my evolutionary algorithm. The cluster runs the job scheduling system OpenPBS[1] that controls the access to computer resources and programs. For communication among processes, Simdist uses the Message Passing Interface[2] (MPI).

D.1 Interactively running a job

To submit a job to the scheduling system's queue, the command `qsub` ("queue submit") is used. This usually reads a shell script with the commands to be run and additional commands to the scheduling system. This mode of operation is described in section D.2 on the facing page, but it is also possible to run a job interactively, which is good for testing purposes as the feedback is immediate.

An interactive job is started by supplying the *-I* parameter to `qsub` and specify a list of resources after a `-l` switch:

```
qsub -I -l nodes=1:ppn=4
```

This command asks for four processors (one node and four processors per node). The command will wait until the requested resources are ready and then spawn a new shell where the jobs can be run. To run a job using these four nodes, the following command can then be used:

[1] http://www.mcs.anl.gov/research/projects/openpbs/
[2] http://www.mcs.anl.gov/research/projects/mpi/

```
mpirun -np 4 -machinefile $PBS_NODEFILE simdist-mpi \
--master =''master_program --param1=value --param2=value''
\
--slave=''slave_program --param1=value --param2=value''
```

where `master_program` is the master EA program and `slave` is the program evaluating the genotypes. Simdist will then use one node for the master program and spread the slaves to the other nodes. For optimal performance, there should be one slave per genome.

D.2 Queueing a job

To queue a job for execution a job script needs to be written and submitted to the scheduler. This script sets up parameters for the scheduler and then runs the command(s) that will do the work. An example script is shown in Fig. D.1 on the next page.

This script file `my_job.job` can be added to the queue by passing the file name as an argument to `qsub`:

```
qsub my_job.job
```

If requested, an email will be sent when the job is begun and finished. The current status of the queue can be viewed by issuing the command `qstat`.

```
#!/bin/sh

# name of the job
#PBS -N phyloea

# max running time hours:minutes:seconds
#PBS -l walltime=03:00:00

# number of nodes:cores per node
#PBS -l nodes=7:ppn=12

# queue to submit job to
#PBS -q default

# send an email on job Abort, Begin, End
#PBS -m abe

# commands to be run
cd $PBS_O_WORKDIR
mpirun -np 84 -machinefile $PBS_NODEFILE \
  simdist-mpi \
  --master-program $PBS_O_WORKDIR/phyloea/master \
  --master-arguments "80 4 300 0.1 0.7" \
  --slave-program $PBS_O_WORKDIR/phyloea/fitness.py
```

Fig. D.1: A job file for running a job on 84 cores (7 nodes, 12 cores on each node). The lines starting with `#PBS` are commands for the scheduling system. The first line specifies which shell should run the job.

Appendix E

Preparing the Sequence Data

The input data for the EA system is assumed to be aligned and without gaps. When using PHYML to evaluate the likelihood of a tree, the input files should be in the PHYLIP format, while the built-in fitness evaluator wants its input in FASTA format. The commands below is a step-by-step recipe to convert a set of FASTA files (a format that is usually available from sequence databases) to one file of aligned sequences without gap sites (see 2.1.2).

1. Concatenate these into one file containing all sequences

   ```
   cat *.fasta > all_sequences.fasta
   ```

2. Align these sequences using `clustalw`[1]:

   ```
   clustalw all_sequences.fasta -ALIGN -OUTPUT=FASTA
   ```

3. Remove gap sites with the `trimAl`[2] tool:

   ```
   trimal -nogaps -in all_sequences.fasta > nogaps.fasta
   ```

`clustalw` can also be used to convert the data to PHYLIP format:

- `clustalw nogaps.fasta -convert -OUTPUT=PHYLIP`

This will write the file `nogaps.phy`.

[1] This can also be done online at The European Bioinformatics Institute where one can submit Clustal alignment jobs and get the results by email: http://www.ebi.ac.uk/Tools/msa/clustalw2/

[2] http://trimal.cgenomics.org/downloads