Heidelberg University
Institute for Computer Science
Database Systems Research Group

Master's thesis

Algorithms for Model Assignment in Multi-Gene Phylogenetics

Name:	Jörg Hauser
E-mail:	joerhau@gmail.com
Matriculation Number:	2848153
Supervisors:	Prof. Dr. Michael Gertz
	Prof. Dr. Alexandros Stamatakis
Date:	August 31, 2012

Ich versichere, dass ich diese Master-Arbeit selbstständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.

Abgabedatum: August 31, 2012

Zusammenfassung

Die Maximum-Likelihood-Methode ist ein beliebter Ansatz um Evolutionsbäume aus genetischen Daten zu rekonstrukieren. Zu diesem Zweck werden stochastische Modelle angewendet, welche die Häufigkeiten von Mutationen spezifizieren. Je nach Wahl des Modells, kann die Plausibilität eines Baumes unterschiedlich sein. Deshalb sollten Modellezuweisungen anhand von angemessen und objektiven Kriterien getroffen werden. Häufig werden allen Partitionen in Multi-Gene Analysen das gleiche Modell zugewiesen. Allerdings ermöglicht die Verwendung eines gemeinsamen Models keine genspezifischen Variationen.

In dieser Arbeit wird das kombinatorische Optimierungsproblem der Proteinmodellzuordnung vorgestellt. Dieses beschreibt die Aufgabe unterschiedliche Protein Modelle verschiedenen genetischen Partitionen zuzuweisen. Angestrebt wird ein Modell für jede Partition, welches den Likelihoodscore maximiert. Die Komplexität der Proteinmodellzuordnung wächst exponentiell mit der Anzahl der in den daten enthaltenen Gene. Daher scheint eine vollständige Evaluaierung aller Modell-Partitionszuordnungen nicht machbar.

Wegen der schnell wachsenden Verfügbarkeit von Daten für komplette Genome, ist es wichtig effiziente Methoden bereitzustellen um geeignete Modellzuweisungen treffen zu können. In dieser Arbeit, werden verschiedene Algorithmen für solche Modellzuweisungen entwickeln und evaluiert.

Abstract

The maximum likelihood method is a frequently applied approach to infer phylogenetic (evolutionary) trees from genetic data. For this purpose stochastic models are applied that specify the rates at which mutations occur. Depending on the model-choice, the plausibility of phylogenies can be different. Therefore, one wants to assign models reasonably. Frequently only one common model is used for distinct genetic partitions for multi-gene phylogenetic inferences. However, this does not allow for gene-specific variation of rates.

In this thesis we introduce the combinatorial optimization problem of protein model assignment. This problem describes the task of appropriately specifying different protein models to distinct genetic partitions. That is, we strive to find a model for each partition such that the likelihood score is maximized. The complexity of this task grows exponentially with the number of genes included in the data. Therefore, it does not to seem to be feasible to exhaustively evaluate all model to partition assignments.

However, it is important to provide objective approaches to quickly determine appropriate model assignments, because of the vastly growing availability of data for whole genomes. Therefore, we also develop and evaluate heuristics to estimate partition distinct model assignments for the use in multi-gene phylogenetics.

Contents

1 Introduction

This introductory chapter provides the motivation for research in the field of phylogenetics. It also describes the basic context and contribution of this work.

1.1 Context and Motivation

Humans have been interested in disentangling the origin of life for a long time. As early as 1859, Darwin published his famous evolutionary theory of natural selection [19]. He introduced the idea of generations of species evolving according to a branching paradigm, thereby, resulting in the observable diversity of life. Furthermore, Darwin is known as the first to have sketched an evolutionary tree. Since that time, many methods to reconstruct evolution have been developed.

The task of determining the evolutionary relationships among species is called *phylogenetics*. Usually knowledge of evolution is summarized in *phylogenetic trees* (also called phylogenies). Figure 1.1 provides a phylogenetic tree for the relationship of mankind and its closest relatives (apes). Entities (nodes) in phylogenies can be entire species (e.g., humans and apes evolved from a common ancestor), groups of species (e.g., hominidae) or even individuals (e.g., this person is an ancestor of that person). Leaf-nodes (tips) are referred to by the term *taxon* (or plural taxa). Within this thesis taxa and species are used synonymously. Inner nodes of trees represent hypothetic common ancestors, which are unknown (they are extinct and thus not observable anymore). Branches (edges) of a phylogeny correspond to evolutionary distance among the connected nodes (e.g., time or divergence). If comparatively high divergence is expected, the edges are longer. For the tree depicted in Figure 1.1, no branch lengths are shown. Usually a strictly bifurcating (complete binary) tree structure is assumed. Hence, every inner node has degree 3. Phylogenies can either be rooted, at the most recent common ancestor of all tips, or unrooted (i.e., not making any statement about the root).

Traditionally, the existence of the three domains of life *bacteria*, *archaea* ("ancient" bacteria), and *eukaryotes* (organisms with complex cell structures) is broadly accepted,

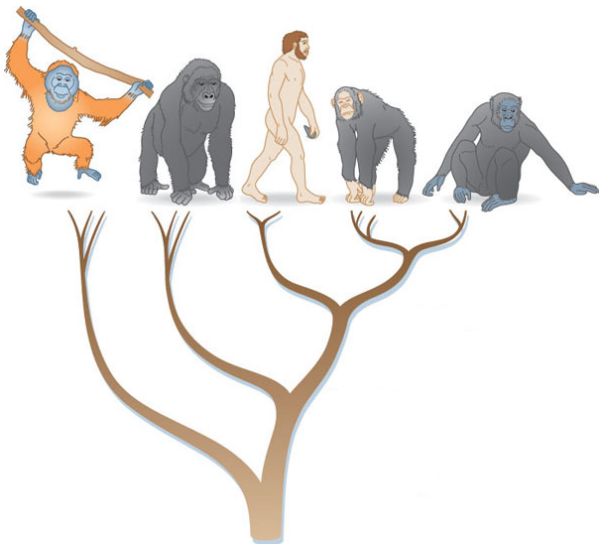


Figure 1.1: Relationship of mankind and apes as estimated by Purvis in [60]. From left to right, orangutan, gorilla, human, bonobo and chimpanzee. *Source: [53]*

based on the work of Woese and Fox in [80]. Recently Boyer *et al.* argued in [10] that the existence of an additional 4^{th} domain may be plausible, namely *Nucleocytoplasmic Large DNA Viruses (NCLDV)*. Figure 1.2 illustrates the four domains. Usually, the goal is to organize members of only one domain in phylogenetic trees. Reconstructing the comprehensive (containing all known living species) tree of life (see `http://tolweb.org/`) is one of the major challenges of the 21^{st} century.

Besides the pure scientific motivation for research in phylogenetics, there are practical aspects, too. For example, methods to improve knowledge about the evolution of organisms help in drug development (see [8]). This is also the reason why viral evolution has received considerable attention. Viruses evolve fast compared to most other organisms, so that they diversify rapidly. Commonly, there are too many drugs to test against a new virus. If we knew which known virus mutated into the new one, health professionals could focus on testing specific drugs that cured the old one. This can speed up the drug development process and also decrease costs.

Every living organism contains genetic information. The amount and its distribution along the genome depends on the species. The term *genome* refers to the entirety of the genetic information of an individual organism. *Genes*, in contrast, are usually thought to represent a specific trait of the organism, such as the hair color. Thus, genes are specific parts of the genome. In practice, the definition of a gene is debatable (see [54]). There even exists an entire scientific area called *genetics* that investigates the properties of genes. To abstract from this question, we interpret genes as meaningful *partitions*

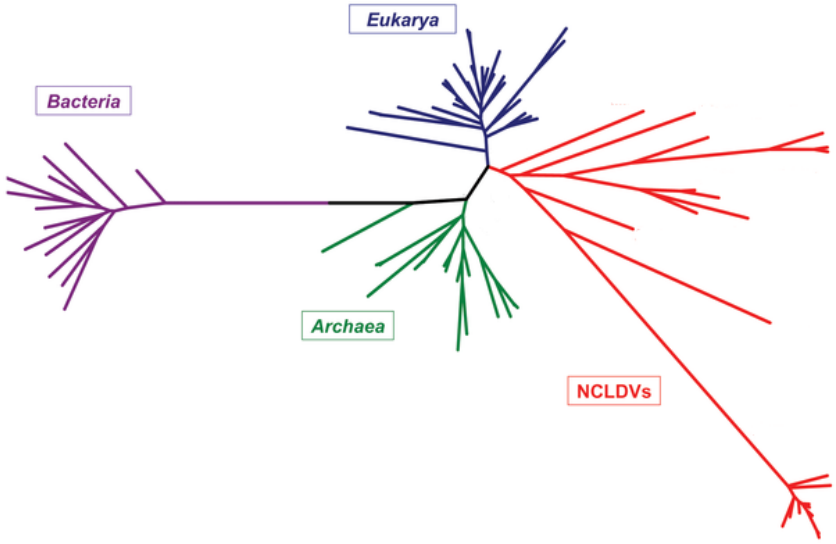


Figure 1.2: Domains of life, assuming that there are four domains. *Source: [10]*

of the genome. Therefore, genes and gene boundaries are given. The terms gene and partition are used interchangeably.

Historically, evolutionary trees were inferred *morphologically*. In morphological phylogenetics visually observable properties, that is, shape or behavior of the organisms, served for reconstructing evolution. Nowadays, computational data-driven phylogenetic methods are predominant. Usually, these methods use genetic data (i.e., molecular information). Therefore, they are of outstanding importance especially for microorganisms, because comparison on morphological properties is difficult for such small organisms. Computational molecular phylogenetics use differences (mutations) in deoxyribonucleic acid (DNA, i.e., molecules composed of nucleotides) or protein (i.e., molecules composed of amino acids) data to compute evolutionary trees. Correlations as well as differences between computational and morphological phylogenetic methods have been investigated by Renaud *et al.* in [64]. Although both approaches are valuable from a biologist's point of view, only computational molecular phylogenetics (also called phylogenomics when multi-gene or whole-genome data is used) will be discussed in this work.

1.2 Challenges

Sequencing techniques are improving rapidly and reduce laboratory effort and cost to retrieve genetic information. Recently, even a USB stick that can sequence genetic data in seconds was presented in [29]. In the past decades, the amount of genetic data in

biological databases like GenBank (`http://www.ncbi.nlm.nih.gov/Genbank/`) or the EMBL Nucleotide Sequence Database (`http://www.ebi.ac.uk/embl/`) have drastically increased. At the same time, the number of species for which whole genomes are available is growing. Therefore, in an increasing number of studies, multi-gene (i.e., concatenated genes) data are used to infer phylogenies.

There exist several computational methods to infer evolutionary trees from molecular data. Two of them (Maximum likelihood (ML) and Bayesian methods of phylogenetic inference) are widely accepted and tend to yield best overall results. Both rely on the usage of flexible and explicit stochastic evolutionary models. These models specify the rate (and therefore the probability) at which mutations occur. For DNA data the rates can be adjusted to the data at hand. Similar adaptations are more difficult for protein data (because there are more amino acid states than deoxyribonucleic acid states, see Section 2.1). Consequently, mutation rates are usually fix for protein models. Thus, with a specific model all rates are given, and exactly the same stochastic process is assumed to describe different data. In other words, distinct DNA models specify a different amount of freedom, whereas distinct protein models fix mutation rates at different values.

One important aspect, which motivates the research in this thesis on multi-gene model assignment, is that distinct genes can evolve under distinct stochastic models. For example, some genes evolve slowly (encounter low mutation rates) and some fast. Phylogenies inferred from distinct genes may show different topologies, that is, the evolutionary distance between species depends on the gene under study. DNA models account for this by using a single model, because mutation rates can be adjusted for every partition separately (using a single DNA model, only the "amount of freedom", not the rates are specified). As protein models have fixed mutation rates, the model usage can considerably influence resulting phylogenies. Therefore, it is important to suitably choose the model for every gene. Unfortunately, it is unknown which genes tend to mutate according to which protein model under which environmental circumstances. Therefore, all of them should be evaluated.

As there are plenty of protein models (each specifying a distinct set of constant rates), and their number is increasing, it is difficult to justify the actual model choice. Furthermore, the difficulty of this task grows exponentially with the number of genes included in phylogenetic analyses. Since every assignment of models to partitions (model assignment) may yields a different phylogeny, ideally every combination should be evaluated. However, there exist m^n distinct assignments of m models to n genes. Thus, it does it is not feasible to evaluate every assignment for large n, because a polynomial time algorithm does not seem to exist.

1.3 Contribution

This work aims at providing efficient methods to obtain suitable model assignments for multi-gene phylogenetic analyses. Because of the exponential growth of the number of assignments, heuristics must be applied to search for suitable model assignments. Although, heuristics help to deal with the immense number of distinct assignments, they suffer from the computationally expensive evaluation of a single assignment. Furthermore, they do not guarantee to find the best solution. However, they provide a feasible approach to find a "good enough" model assignment.

To develop heuristic strategies we introduce the protein model assignment (PMA) problem (i.e., the problem of adequately assigning models to multi-gene protein data) in terms of a combinatorial optimization problem. The PMA problem can be important for many protein-bases phylogenomic analysis. To search for optimal PMA we adapt deterministic and randomized search heuristics and assess promising combinations of these self-contained heuristics to improve their performance. Furthermore, we develop algorithmic optimizations by including domain specific knowledge. These optimizations are able to significantly decrease the execution time of the heuristics. In particular we present a clustering approach, to guide heuristics to high quality model assignments. Furthermore, we exploit changes to the evaluation of the likelihood score to reduce the runtime of included numerical optimization algorithms and apply an archive to provide the scores of already evaluated assignments. These changes do not impact the result quality of the heuristics. Lastly, we develop a strategy to approximate the potential of an assignment (i.e., to pre-score assignments before actually evaluating them). This strategy can, however, impact the result quality negatively because it prunes major parts from the search-space. Finally, we evaluate our heuristics and improvement strategies on real and synthetic data to assess their usefulness.

1.4 Structure of the Thesis

The thesis is structured as follows. First, the basics of molecular phylogenetic inference are briefly presented in Chapter 2. Chapter 3 reviews related work on model-selection. In Chapter 4 we focus on establishing various heuristics for model assignment. Thereafter, we develop algorithmic optimizations to reduce computational requirements in Chapter 5. The evaluation of the proposed heuristics and algorithmic optimizations takes place in Chapter 6. Conclusions and an outlook are provided in Chapter 7.

2 Computational Molecular Phylogenetics

This chapter describes the basics of computational phylogenetic inference. The goal is to provide the necessary background knowledge for non-biologists. In [83] Yang provides a comprehensive overview of computational molecular phylogenetic methods.

Initially, (Section 2.1) the basic concepts and goals will be explained. Section 2.2 describes the necessary preprocessing task of sequence alignment, which is required for most phylogenetic analyses. In Section 2.3 we will take a closer look at phylogenetic inference methods. The most important concept for this thesis—protein substitution models—will be explained in Section 2.4.

2.1 Principles

Computational molecular phylogenetic approaches infer evolutionary relationship from genetic information, that is, molecular (usually DNA or protein) data. In this work only the computational aspects are covered. Therefore, it is sufficient to consider genetic information to be sequences of DNA bases or amino acids. To abstract from chemical properties, sequences are represented as character strings. The characters stand for the molecules' components. For DNA `A`,`C`,`G`, and `T` represent adenine, cytosine, guanine, and thymine (i.e., bases of DNA or nucleotides). For proteins there are twenty amino acids. Their usual character-mapping is based on [15] and shown in Table 2.1.

Differences in the sequences of distinct organisms are used to reconstruct evolutionary histories. The differences are a result of evolution and express mutations of molecules. One differentiates mutations that can be observed in the data from others which may have happened, but are not observable anymore (e.g., because they were disadvantageous and have been "thrown away" by selection, or because they were intermediate mutations only). Observable mutations are called *point accepted mutations*.

Amino Acid	1-Letter
Alanine	A
Arginine	R
Asparagine	N
Aspartic acid	D
Cysteine	C
Glutamic acid	E
Glutamine	Q
Glycine	G
Histidine	H
Isoleucine	I

Amino Acid	1-Letter
Leucine	L
Lysine	K
Methionine	M
Phenylalanine	F
Proline	P
Serine	S
Threonine	T
Tryptophan	W
Tyrosine	Y
Valine	V

Table 2.1: Amino acids and their abbreviations according to [15].

Phylogenetic inference methods can be classified into *supertree* and *supermatrix* approaches (see [21]). Supertree methods aim at analyzing information of distinct genes and organisms separately (by creating multiple phylogenies). The resulting phylogenies are merged together afterwards into a comprehensive supertree. The supertree method used to be the only feasible approach to reconstruct large phylogenies. Supermatrix approaches, in contrast, aim at analyzing all available data of all organisms simultaneously (thus, generating only one phylogeny—the supermatrix tree). One expects an increasing evolutionary signal by the concatenation of genes of many taxa, as well as a decrease of noise. This is a major advantage of supermatrix over supertree approaches.

2.2 Multiple Sequence Alignment

Molecular evolution consists of substitutions (A $\rightarrow$ R), insertions (AI $\rightarrow$ ACI) and deletions (ACI $\rightarrow$ AI) of characters in sequences. Insertions and deletions are often termed *indels*.

Because of indels, sequences of different organisms do not necessarily have the same length. In order to compare two molecular sequences of different length, a sequence alignment must be computed first. For example, consider two species with corresponding molecular sequences S_1 and S_2:

S_1: N A D A A I
S_2: N A A Q I

One could align these sequences as follows:

S_1: N A D A A I
S_2: N A - A Q I

Insertion and deletion are complementary events. Depending on whether S_1 evolved from S_2, or the other way around, D was either inserted or deleted.

The alignment task involves the minimization of a score function:

$$F_{score} : S_1 \times S_2 \rightarrow \mathbb{R}$$

F_{score} describes the distance of S_1 to S_2 in a given alignment (the larger the value, the more different the sequences are). For the example at hand, this function can be defined as:

$$F_{score} = \sum_i map(S_{1_i}, S_{2_i})$$

with S_{1_i} and S_{2_i} denoting the i^{th} character of sequence S_1 and S_2, and a character mapping function:

$$map(S_{1_i}, S_{2_i}) = \begin{cases} 2 & \text{if } S_{1_i} = \text{-} \vee S_{2_i} = \text{-} \quad \text{(indel/gap)} \\ 1 & \text{if } S_{1_i} \neq S_{2_i} \quad \text{(substitution)} \\ 0 & \text{else} \quad \text{(nothing happend)} \end{cases}$$

This way matching characters are rewarded and gaps are penalized. For the alignment above, the value of the scoring function is 3 (as there is one indel and one substitution).

Mostly, the overall goal of sequence alignment is to generate two strings of the same length with as many matching characters for the positions as possible by inserting gaps. Usually this is done by the application of dynamic programming techniques (see e.g. Needleman-Wunsch algorithm in [50]).

For more than two sequences, such a mapping is called a *multiple sequence alignment*. The amount of sequences can range from a few species to several thousands. Usually, for multiple sequence alignments the scoring function is defined as the sum of all pairwise scores. For n species this is:

$$F_{score} = F_{11} + \cdots + F_{1n} + \cdots + F_{nn}$$

Generating optimal multiple sequence alignments is NP-hard (see [78]). There exist various approaches to speed up alignment generation and to improve the quality of the results. The quality of an alignment is of major importance for phylogenetic analyses, since it is used as input. Therefore, frequently tree quality depends on alignment quality.

2.3 Phylogenetic Inference

In phylogenetics we search for the best phylogeny given a multiple sequence alignment. We outline the necessary tasks for this purpose in this section.

In Section 2.3.1 we outline the search-space (the set of possible phylogenies for a given set of taxa) . Thereafter, two methods to score phylogenies are described in Section 2.3.2. Finally we present two algorithms to search for the most plausible tree (Section 2.3.3).

2.3.1 Tree-Space

For phylogenetic inference the size of the tree-space is of great importance. Figures 2.1 and 2.2 depict all possible rooted and unrooted binary tree topologies for small sets of three and four species, respectively. One can easily observe that the number of different topologies for rooted trees is larger than that for unrooted binary trees. This holds for all tree sizes with $n \geq 3$ species. Table 2.2 lists the numbers of possible trees for some

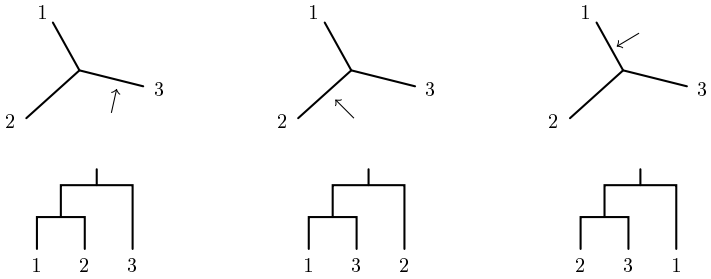


Figure 2.1: Possible rooted trees and the placement of the root in the corresponding unrooted tree for three taxa. Arrows in the upper tree denote the placement of the root within the lower rooted tree. There is only one unrooted tree topology for three organisms. *Source: [55]*

values of n. According to Cavalli-Sforza and Edwards (see [13]) for n species there exist

$$3 \cdot 5 \cdot 7 \cdots (2n-3) = \prod_{i=3}^{n}(2i-3) = \frac{(2n-3)!}{2^{n-2}(n-2)!}$$

different rooted binary trees. This number decreases slightly to

$$\prod_{i=3}^{n}(2i-5)$$

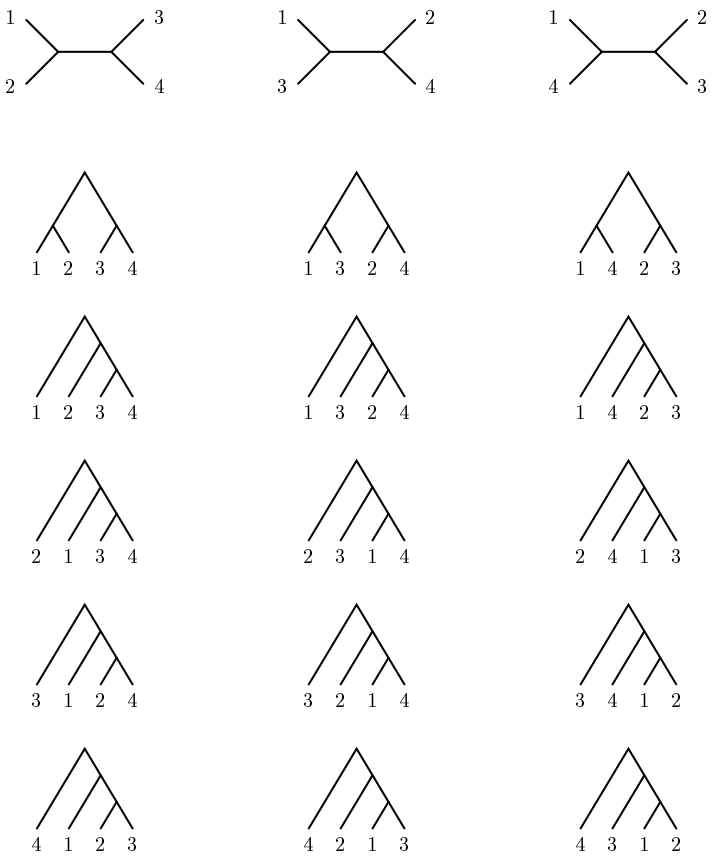


Figure 2.2: Possible rooted and unrooted trees for four taxa. The topmost trees depict the three possible unrooted topologies. Their rooted equivalents are listed within the corresponding columns. *Source: [55]*

for unrooted trees. Usually, no data about the root is available, and hence an unrooted tree is assumed.

In practice, it is not feasible to evaluate all possible tree-topologies. Therefore heuristics must be applied in order to explore the tree-space and find "good" topologies. The heuristics used in the context of this work are described in Section 2.3.3. Before this, we describe methods to evaluate the plausibility of a given phylogeny in the next section.

2.3.2 Conceptual Models

Phylogenetic tree inference always relies on assumptions. Assume that we know two "good" phylogenies of some species. In order to determine which tree is more plausible, one hast to make hypotheses about the evolutionary process. Analogous to Kelchner and Thomas in [40], we refer to these hypotheses by the term *conceptual model*. Commonly, conceptual models imply an *optimality criterion* according to which the plausibility of a tree can be assessed. In the following we outline two popular optimality criteria,

n	No. rooted trees	No. unrooted trees
2	1	1
3	3	1
4	15	3
5	105	15
6	945	105
7	10395	945
8	135135	10395
9	2027025	135135
10	34489707	2027025
50	2.8×10^{76}	3×10^{74}

Table 2.2: Number of rooted and unrooted trees as a function of the number of taxa n. *Source: [55]*

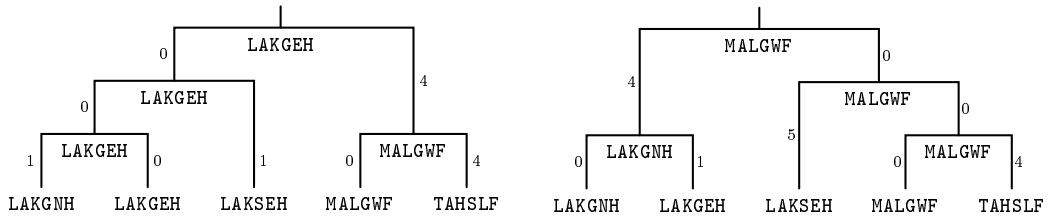
which are used in the context of this thesis—the *Maximum Parsimony Criterion* and the *Maximum Likelihood Criterion*.

Maximum Parsimony

The Maximum Parsimony (MP) criterion assumes that the evolution of sequences is best explained by the tree with least amount of substitutions. In other words, evolution is parsimonious. Thus, the tree with the least number of substitutions is considered to be the one that best explains their evolutionary history. Consequently, the goal is to minimize the sum of all pairwise (parent $\leftrightarrow$ child) Hamming distances ([30]) in the tree. Put differently, the goal is to minimize the number of positions at which the corresponding states of parents and their children are different. The approach was described by Edwards and Sforza in [23].

The number of substitutions necessary to construct the observed data for a specific tree is called *total score* or *tree length*. To compute the total score of a tree, we sum over the scores of all alignment sites. Sites that have identical nucleotides for all species can be neglected as they have zero cost for all possible trees. They are called uninformative sites. Also sites with less than two different states for more than one taxon per state are uninformative. Figure 2.3 shows two exemplary trees and their respective parsimony score for the following alignment containing organisms $S_{1...5}$

S_1: L A K G N H
S_2: L A K G E H
S_3: L A K S E H
S_4: M A L G W F
S_5: T A H S L F



Total cost = 10 Total cost = 14

Figure 2.3: Maximum Parsimony cost computation. *Adapted from [55].*

As already mentioned in the introductory chapter (Section 1.1), the sequences of inner nodes are unknown. Therefore, they are assigned a candidate state per site that minimizes the score. In order to be able to compute an optimal state the tree must be rooted. Sets of candidate states can be generated bottom up via a post-order tree traversal. Figure 2.4 shows the generation of candidate states that minimize the cost for one site. The states at the leafs (N, E, W, L), which are known, serve as candidate states ({N, E}, {N, E}, {W, L}) for the ancestral nodes. This principle is continued until the root {N, E, W, L} is reached. Note that the same method can be applied for the remaining alignment sites.

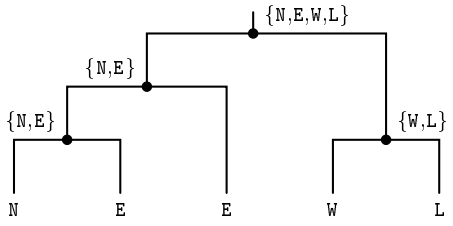


Figure 2.4: Candidate generation for inner nodes. Shown for site five of the left tree of Figure 2.3. *Adapted from [71].*

While MP is useful for scenarios where organisms evolved through a few mutations, it can fail for because of long branch attraction (see [55]).

Maximum Likelihood

With the Maximum Likelihood (ML) criterion, Felsenstein introduced a probabilistic conceptual model based on Markov chains in [25]. ML represents a computationally feasible stochastic approach. ML works better than parsimony if sequences evolved with many mutations (long branches).

The basic idea is to score candidate trees according to their probability to generate the observed data ($P(data|tree)$). The phylogeny that yields the highest value is assumed to be the "best" one. ML (like the Maximum Parsimony criterion) assumes independent evolution among sites (i.e., the evolution of one site does not influence its neighbors). Although this is a restrictive and unrealistic assumption from a biological perspective (especially with respect to insertions and deletions), it is necessary for computational reasons (see [25]). The main differences to MP are that minimal evolution is not assumed and that substitution pairs can be scored differently. Accordingly, a major advantage is the specification of explicit probabilistic models of evolution.

$P(data|tree)$ is usually called the likelihood of the tree. It is important to note, that the likelihood of a tree is not the probability of the tree to be the correct one ($P(data|tree) \neq P(tree|data)$). This is why the likelihood values of all possible trees do not sum to 1.0. Recently Bayesian approaches for phylogenetic inference have also been introduced in [84]. They compute the posterior probability of the data producing the tree ($P(tree|data)$). That is, the probabilities of all phylogenies sum to 1.0.

The likelihood is evaluated via a Markov process that is computed by a post-order traversal of the tree. As common for Markov processes, transitions between states are independent from their history. That is, the probability of being in state j at time $t + \delta$, only depends on the probability of being in state i at time t and the transition probability $P_{ij}(\delta)$. Hence, the following two ingredients are necessary:

1. prior/initial state probabilities π_i. The probabilities of being in each specific state i initially
2. transition probabilities $P_{ij}(\delta)$. The probability of changing from state i to state j within time δ

For the likelihood computation, a rooted tree is necessary. The computation is outlined on the tree given in Figure 2.5. The derived formula can easily be extended to any other tree. Because sites are assumed to be independent, one site suffices to explain the principle. The overall likelihood can be computed by multiplying the per-site results.

Figure 2.5 shows a rooted evolutionary tree for three known organisms. Their states (e.g., observed amino acids) are denoted as S_1, S_2 and S_3. The length of the branches

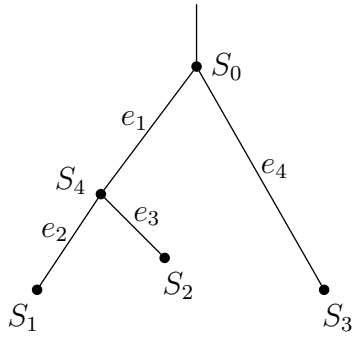


Figure 2.5: Simple maximum likelihood tree. *Adapted from [25].*

are referenced by $e_{1...4}$. Again, the states of the inner nodes are not known. If they were known, the likelihood L of the tree could be computed by following the Markov process from the root node S_0 to the tips ($S_{1...3}$):

$$L = \pi_{S_0} \cdot P_{S_0S_4}(e_1) \cdot P_{S_4S_1}(e_2) \cdot P_{S_4S_2}(e_3) \cdot P_{S_0S_3}(e_4)$$

That is, the likelihood is the product of the prior probability π_{S_0} of state S_0 times the probabilities at each edge. As the inner states are unknown, S_0 and S_4 could be assigned any state maximizing the likelihood. That is, the score is the sum of the former formula over all possible states for S_0 and S_4:

$$L = \sum_{S_0} \sum_{S_4} \pi_{S_0} \cdot P_{S_0S_4}(e_1) \cdot P_{S_4S_1}(e_2) \cdot P_{S_4S_2}(e_3) \cdot P_{S_0S_3}(e_4)$$

In analogy to [25, 71] this translates into:

$$L = \sum_{S_0} \pi_{S_0} \cdot \underbrace{\sum_{S_4} \left(P_{S_0S_4}(e_1) \cdot \underbrace{P_{S_4S_1}(e_2)}_{S_1 \text{ subtree}} \cdot \underbrace{P_{S_4S_2}(e_3)}_{S_2 \text{ subtree}} \right)}_{S_4 \text{ subtree}} \cdot \underbrace{P_{S_0S_3}(e_4)}_{S_3 \text{ subtree}}$$

The former equation highlights that the likelihood can be evaluated bottom-up starting at the leafs. Thus, the computation is usually implemented by a postorder tree-traversal. For a numerical example of the likelihood computation for DNA data see [83].

Usually, likelihood values are extremely small. Therefore, it is common to compute the logarithm of the likelihood score—the log likelihood (lnL)—instead. Because the logarithm is monotonic this does not impact the order of likelihood scores.

For simplicity, we just fixed the branches (e_1, e_2, and e_3) until now. To obtain the ML

score of a tree, the branch lengths must be adjusted to their optimum value. Mostly, evolution is assumed to be time-reversible. That is, the evolutionary process is identical if followed forward or backward in time:

$$\pi_i P_{ij}(\delta) = P_{ji}(\delta)\pi_j$$

An important consequence of time reversibility is the so-called *pulley principle* (see [26, 83]). We already mentioned that a rooted tree is necessary for the likelihood computation. Luckily, because of the pulley principle, the likelihood score does not depend on the placement of the root. Therefore, the root can be moved all-over the tree, without affecting the likelihood (as long as the branch lengths remain fixed).

To optimize the branches, the pulley principle is valuable, too. Initially, the branches are assigned a "reasonable" default value. These values are optimized for each branch independently by iteratively placing a virtual root into them. The pulley principle avoids that the likelihood needs to be re-evaluated for the whole tree each time a single branch is changed. In order to optimize a single branch, its length is adjusted to the value resulting in the highest likelihood score. In [26], Felsenstein and Churchill, proposed to use the Newton-Raphson method for this purpose.

2.3.3 Heuristic Tree-Search

In Practice, both MP and ML, suffer from the immense tree-space. A priori every possible topology can be considered. Therefore, a simple approach could exhaustively evaluate all trees and pick the most "plausible" one (exhaustive search). Because of the large number of unrooted trees, an exhaustive search is not feasible for common optimality criteria. Consequently, heuristics to efficiently infer "good enough" phylogenies are important. Many different heuristics have been developed and implemented in various software packages. For a comprehensive list, see Joe Felsenstein's website at `http://evolution.genetics.washington.edu/phylip/software.html`.

RAxML (Randomized Accelerated Maximum Likelihood) is the acronym of a software for Maximum Likelihood based phylogenetic inference, which was presented by Stamatakis *et al.* in [72]. RAxML-Light (see [74]), a modification of RAxML, was chosen as the basis to develop and evaluate the approaches described in Chapter 4 and following, because it is especially designed for large data-sets. Therefore, we rely on the heuristics implemented in RAxML and RAxML-Light (see [72]). For MP the *randomized stepwise addition algorithm* is used, whereas the ML tree search relies on *lazy subtree rearrangements*. Both are briefly outlined in the following.

Randomized Stepwise Addition: is a greedy heuristic. Its basic idea is to insert, new, randomly chosen taxa at the best position in a given tree. For this purpose, first a set of three random organisms is chosen to build an unrooted tree. Afterwards, the remaining organisms are inserted one-by-one at the best-scoring position in random order. Depending on the insertion order, results can look differently every time the method is invoked. This is acceptable, first because phylogenetic analyses are usually executed multiple times, and second, because parsimony trees are only used as starting points of the ML search only here (i.e., the tree topology is refined at a later point).

Lazy Subtree Rearrangement: uses a comprehensive starting tree (e.g., based on the randomized stepwise addition algorithm). This is mainly due to close relationship of Maximum Parsimony and Maximum Likelihood inference, especially for simple evolutionary models (see [72]).

Based on this starting tree (which will serve for the methods proposed in this thesis as well) the ML score is optimized by pruning and reinserting all subtrees of the starting tree. Subtrees are pruned and reinserted in-between nodes of range $r \in \mathbb{N}$, where r describes the number of nodes separating the pruning and the reinsertion branch. r is called the *rearrangement setting*. This strategy is repeated for the resulting phylogenies until no better tree can be found.

2.4 Models of Evolution

The Maximum Likelihood criterion includes an explicit, stochastic model of character evolution. This model is interchangeable without affecting the basic idea of ML. In this section we explain how the selected model influences the computation of the likelihood score and the tree.

We start by presenting the ingredients of a model in Section 2.4.1. Thereafter, we explain the difference between empirical and mechanistic models (Section 2.4.2). In Section 2.4.3 the most common models for protein data are presented. Finally, we outline commonly applied approaches to further enhance the models in Section 2.4.4.

2.4.1 Transition-Probability Matrix

A model of molecular substitution specifies the transition probabilities ($P_{ij}(\delta)$) of the Markov process. The ability to explicitly specify the properties of the evolutionary

process provides more the flexibility compared to optimality criteria that have built-in assumptions (e.g., the Maximum Parsimony criterion). DNA models are even more flexible than protein models, because substitution rates can be adjusted to the data at hand. For protein models substitution rates are usually constant, because of the larger number of possible states. This is also the reason why we outline the basics for DNA models. The approaches can, however, be easily extended to proteins (see, for example, [36]).

A model of molecular evolution consists of a matrix Q indicating the rate of every possible substitution for an infinitesimal time period:

$$Q = \begin{array}{c} \begin{array}{cccc} A & C & G & T \end{array} \\ \begin{pmatrix} -a\pi_C - b\pi_G - c\pi_T & a\pi_C & b\pi_G & c\pi_T \\ a\pi_A & -a\pi_A - d\pi_G - e\pi_T & d\pi_G & e\pi_T \\ b\pi_A & d\pi_C & -b\pi_A - d\pi_C - f\pi_T & f\pi_T \\ c\pi_A & e\pi_C & f\pi_G & -c\pi_A - e\pi_C - f\pi_G \end{pmatrix} \end{array}$$

For time-reversibility, the substitution rates $a, \ldots, f \in \mathbb{R}^+$ need to be symmetrical. The base frequencies $\pi_{A\ldots T}$ are restricted to sum up to 1.0. Note that the diagonal entries are defined, such that each row sums up to 0. Hence $-q_{ii}$ gives the substitution rate of state i. Such models are called *general time reversible* (GTR) models.

According to [46], the values of the substitution rate matrix translate into a matrix of transition probabilities for time t as follows:

$$P(t) = e^{Qt}$$

The major difference between DNA and protein evolution models is the size of Q. For DNA (4 states), Q is a 4×4 matrix, whereas for proteins it is a 20×20 matrix. The DNA GTR-model contains 10 parameters (6 substitution rates and 4 base-frequencies). As the rates are relative, f is usually fixed to 1. Moreover, one frequency π_i is given by the remaining π_j $(i \neq j)$ with:

$$\pi_i = 1 - \sum_j \pi_j$$

Therefore, there are 8 free parameters, which must be estimated. The number of free parameters for the GTR model can be given as a function of the number of possible states n as follows:

$$\underbrace{n-1}_{|\text{frequencies}|} + \underbrace{\frac{n^2-n}{2} - 1}_{|\text{exchangeability rates}|}$$

Accordingly, for proteins there are 208 free parameters.

The most simple model assumes equal substitution rates and equal frequencies for all states. According to the work of Jukes and Cantor in [38] this model is called JC. However, in practice, this assumption oversimplifies reality. There are substantial differences between substitution rates. Generally, substitutions between chemically and physically close states are more frequent than mutations between more distinct states. Moreover, the genetic code describes which sets of three bases of DNA (called codons) encode for an amino acid (see [52]). Some amino acids are encoded by one codon and some by two or more. If there is more than one codon representing an amino acid, the codons often only differ by one base. For some amino acid substitutions, two or more bases of the codon must change. These mutations often have comparatively small substitution rates. In addition, environmental stress influences substitution rates, that is, some mutations are disadvantageous. Also secondary structure of proteins can impact substitution rates.

2.4.2 Mechanistic vs. Empirical Models

Models (like the GTR model) that allow for rate estimation using the alignment at hand are usually called *mechanistic models*. For *empirical models*, in contrast, parameters are derived from a set of large, closely-related alignments. That is, the rates are estimated on data that is different from that to be analyzed. If empirical models are applied in an analysis, there are no substitution rates that must be estimated. Usually, mechanistic models are applied to DNA, and empirical models to protein analyses. Therefore, we use the terms *mechanistic* and *DNA* model, as well as *empirical* and *protein* model synonymously.

DNA models include parameters to adjust them to the actual sequences. Mechanistic protein models need to include a comparatively high amount of parameters to specify reasonable rates for all amino acid substitutions. The danger of overfitting and over-parametrization increases with the number of parameters(see [24]). The more free parameters a model has the "better" the likelihood will be. However, this is not necessarily indicative for a better suited model, but probably only the result of higher computational freedom. Moreover, conducting ML estimates of of mechanistic protein models is computationally extremely expensive, because of the large number of amino acid states and free parameters. Therefore, empirical models are mostly used for protein-based ML inferences.

The fixed substitution rates and amino acid frequencies of empirical protein models reduce the number of free parameters from 208 for the GTR-model to 0. This is also

beneficial because empirical models can be applied to small datasets. On the other hand, it is not clear which model is appropriate for which data. Additionally, there is no guarantee that any model is appropriate for the data at hand. However, usually this decreased flexibility is accepted.

2.4.3 Common Protein Models

One of the first empirical protein models was proposed by Dayhoff and Schwartz in [20]. They counted over 1,500 mutations in about 100 nuclear encoded proteins, even though no computer-based alignments were available at this time (1978). Jones *et al.* used the same approach in 1992 for 16,300 protein sequences, counting 59,190 mutations. This model is called JTT. Usually, the JTT model is considered to be more accurate, as it is based on a larger data sample. In general, the more data is used, the better the resulting model is expected to be.

Differences between the models are due to the data type or the estimation methods. Every model is based on a different set of alignments. Some models even focus on specific species or specific genes (e.g., viral species or mitochondrial genes). This leads to substantial differences in substitution rates among models. Dayhoff and Schwartz already proposed two methods for parameter estimation in their initial work. Whelan and Goldman argued that the Dayhoff approach may lead to systematic error in [79]. They proposed a ML based approach to create the WAG model.

Some models have rather similar rates and origin, whereas some others are quite distinct. It is difficult to rationally chose the "best" model for phylogenetic analyses. Frequently, more than one can be appropriate. Obviously, knowledge of the used data and methods to create the model can help to choose the model. We do not comprehensively outline these details, though. However, Table 2.3 references the most common models, which are considered in this thesis.

2.4.4 Model Derivates

Besides the differences in substitution rates, there are three adoptions that can be applied to empirical protein models. These cannot be incorporated into the instantaneous rate matrix Q. However, they can be estimated on the data at hand.

Firstly, Reeves found in [63] that the fit of any model is vastly improved by allowing a proportion of sites to be invariant. Certain sites are unlikely to undergo mutations (e.g., for functional reasons). Although this fact was known before (see [32, 31]), invariant

N	Acronym	Reference	Target area/genes
1	Blosum	Henikoff and Henikoff in 1992 [34]	Nuclear
2	cpREV	Adachi *et al.* in 2000 [5]	Plastid
3	Dayhoff	Dayhoff and Schwartz in 1978 [20]	Nuclear
4	Dayhoff-dcmut	Kosiol and Goldman in 2005 [43]	Nuclear
5	FLU	Cuong *et al.* in 2010 [17]	influenza viruses
6	HIVb	Nickle *et al.* in 2007 [51]	Retroviral
7	HIVw	Nickle *et al.* in 2007 [51]	Retroviral
8	JTT	Jones *et al.* in 1992 [37]	Nuclear
9	JTT-dcmut	Kosiol and Goldman in 2005 [43]	Nuclear
10	LG	Le and Gascuel in 2008 [44]	Nuclear
11	mtART	Abascal *et al.* in 2007 [3]	Mitochondrial
12	mtMAM	Cao *et al.* in 1998 [12]	Mitochondrial
13	mtREV	Adachi and Hasegawa in 1996 [4]	Mitochondrial
14	mtZoa	Rota-Stabelli *et al.* in 2009 [67]	Mitochondrial, animals
15	PMB	Veerassamy *et al.* in 2003 [77]	Nuclear
16	rtREV	Dimmic *et al.* in 2002 [22]	Retroviral
17	VT	Müller and Vingron in 2000 [49]	Nuclear
18	WAG	Whelan and Goldman in 2001 [79]	Nuclear

Table 2.3: Common protein evolution models with initial application area (target-genes).

sites were not taken into account. Frequently, invariable sites contain the same amino acid for all species. Hence, they do not contribute for methods like MP. However, for ML the probability of not observing a substitution is important. Invariant sites are commonly indicated by adding $+I$ to the model name.

Similarly, Yang proposed the idea of among site rate variation (also called *rate heterogeneity*) in [82]. For this purpose, substitution rates among sites are described by a Γ-distribution with shape parameter α. This is also the reason why this option is usually indicated by $+\Gamma$. A small α reflects significantly varying rates among sites, that is, few sites with rapid evolution. A large α, in contrast, suggests minimal among-site variation. Figure 2.6 illustrates the Γ-distribution for three different shape-parameters. For an in-depth explanation see [55].

Commonly, amino acid frequencies (π_i) of empirical protein models are different from the frequencies in the alignment. In [11], Cao *et al.* proposed to use the amino acid frequencies of the data under study. They obtained better phylogenies, especially if the discrepancies were large. Using the frequencies obtained from the data is referred to $+F$. We call this the usage of *empirical base frequencies*. Note that for mechanistic models, frequencies are always estimated on the alignment.

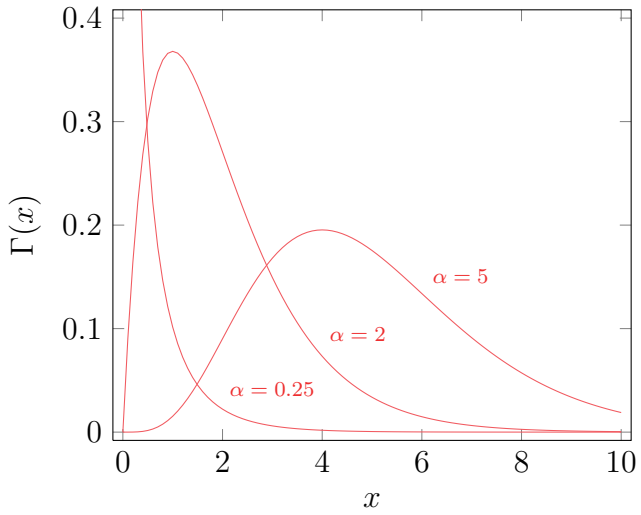


Figure 2.6: The Γ-distribution for different values of α.

Chapter Summary:

This chapter introduced the basic terms and tasks of molecular phylogenetic inference. The principle of multiple sequence alignment (a preprocessing task in most phylogenetic analyses) was presented. Additionally we discussed two criteria (Maximum Parsimony and Maximum Likelihood) to score phylogenetic trees. Both criteria make simplifications, to model the evolutionary process. We call the entirety of these assumptions a conceptual model. Moreover, the large search space of phylogenetic trees was explained. Therefore, even if one is willing to accept the assumptions of the criteria, it is not feasible to find the optimal phylogeny for a large number of species.

One of the major factors influencing the likelihood score are the rates of state substitutions. For DNA these rates are usually estimated for the data at hand, whereas they are mostly derived from fixed data (empirical models) for proteins. There exists a variety of different empirical models of amino acid substitution. However, a model suitable for one dataset (i.e., organism or type of genes) is not necessarily suitable for other datasets. Therefore, it is necessary to objectively specify the applied models for each phylogenetic analysis.

3 Multi-Gene Model Selection

In [13], Cavalli-Sforza and Edwards found that the validity of an inferred phylogeney strongly depends on the correctness of the underlying model. Therefore, it is important to use the most appropriate model. This chapter presents approaches for selecting models for phylogenetic inference.

In Section 3.1 related work on model selection is presented. Afterwards, the model assignment problem for multi-gene datasets is introduced in Section 3.2.

3.1 Related Work and Objectives

Model selection seeks to find the best-fit model given a set of candidate models. Some models can be organized hierarchically from a complex (parameter rich) version (e.g., GTR+Γ+I) to simpler versions (e.g., JC), which are special cases of the complex one. The number of free parameters in this hierarchy decreases. That is, the simpler model can be obtained by fixing some parameters of the complex model. We call such models *nested models*. In general, more complex (parameter-rich) models fit the data better than simpler models. This is not necessarily due to the suitability of the model, but can just be the result of a larger computational freedom. To select the best-fit model, all models and options must be evaluated. This includes parameter estimation for mechanistic models. The challenge is to balance accuracy and simplicity. In other words, use a sufficiently complex model (with as many free parameters as necessary), but not more. The approaches differ slightly for mechanistic and empirical models.

Mechanistic Models: An initial popular strategy for model selection are hierarchical likelihood ratio tests (hLRT) [35]. These carry out multiple pairwise $\mathcal{X}^2$-tests between a specific simple model H_0 (null hypothesis) and a more complex model H_1 (alternative hypothesis). The basic assumption is that, with $L(data|H_0)$ beeing the likelihood under

model H_0, the test statistic δ:

$$\delta = 2ln\frac{L(data|H_0)}{L(data|H_1)} = 2(lnL(data|H_0) - lnL(data|H_1))$$

is $\mathcal{X}^2$ distributed if H_0 is true (with the degree of freedom k equal to the difference of free parameters between H_1 and H_0). If the probability of observing δ's value of is small (usually $\leq 5\%$) under the $\mathcal{X}^2_k$ distribution, H_0 is rejected. That is, the inclusion of more parameters in H_1 significantly increases the likelihood. Otherwise H_0 is accepted.

In [56], Posada and Buckley argue that likelihood ratio tests are not the optimal strategy, because they do not account for non-nested models. Instead, they advocate the use of the Akaike Information Criterion (AIC, see [6])

$$AIC = -2lnL(data|H_i) + 2K_i$$

or the Bayesian Information Criterion (BIC, see [69])

$$BIC = -2lnL(data|H_i) + K_i \cdot log(n)$$

where K_i is the number of free parameters in the i^{th} model (H_i) that yields log likelihood $lnL(data|H_i)$ and n is the sample size[1]. BIC and AIC allow for simultaneous comparison of nested and non-nested models. Both information criteria, as well as hLRT, are implemented for DNA alignments in ModelTest [57]. ModelTest calculates the likelihood score for all models and possible options (+I or +Γ) on a fixed tree. The tree is either given by the user or created using the neighbor joining algorithm (see [28]). Afterwards, models are ranked according to BIC or AIC.

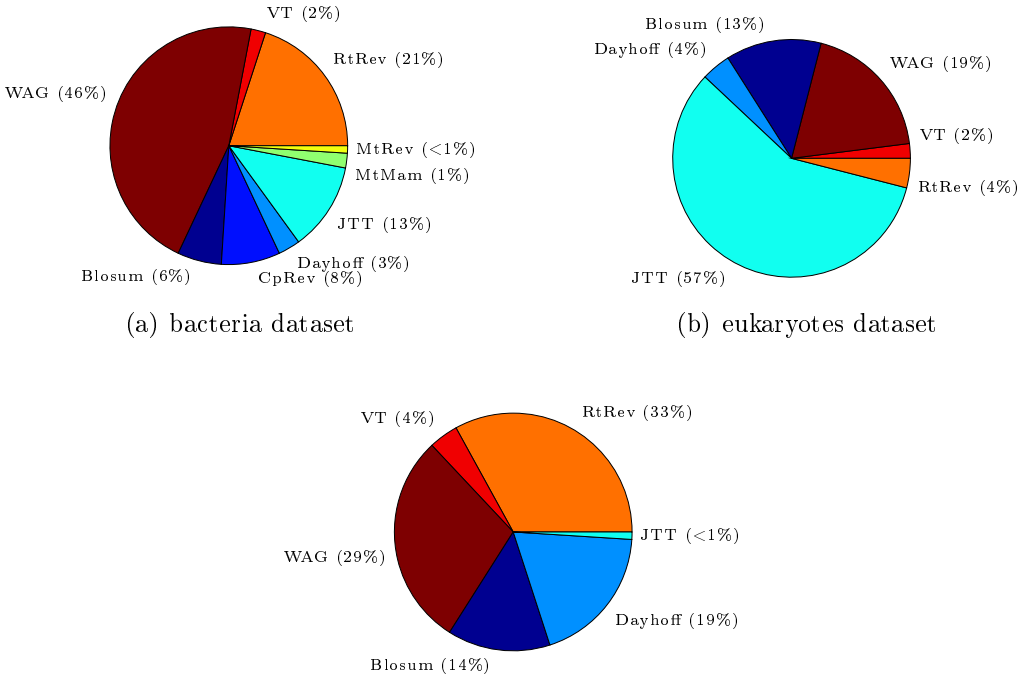
In [65], Ripplinger and Sullivan investigated the effects of model selection on ML tree inference for DNA data. They conducted analyses for 250 phylogenetic datasets and found that AIC typically selects more complex models than BIC or hLRT. Moreover, rate heterogeneity (+Γ option) was included in almost all selected models. The largest variation in the number of free parameters was due to different choices for the instantaneous rate matrix Q.

Empirical Models: For models with fixed substitution rates there do not exist free parameters in Q. Therefore, biologists often carry out an empirical model choice. That is, a model that was estimated on similar data is assumed to suit the data at hand best.

[1] It is not clear what the sample size of a sequence alignment is. In ProtTest [2, 18] by default, the total number of characters of the alignment is used as sample size.

For example, to infer a viral phylogeny one would use the HIVb, HIVw, or FLU models, as all were estimated on viral alignments. However, this approach does not rely on any objective criterion. Moreover, there remain three common virus models plus the options (+I, +Γ, and +F).

Usually, selection of empirical models is done analogously to mechanistic model selection. Abascal *et al.* extended ModelTest to protein models in the ProtTest program ([2]). The major difference is that hLRT are not implemented, since only few of the empirical protein models are nested.



(a) bacteria dataset

(b) eukaryotes dataset

(c) archaea dataset

Figure 3.1: A break-down of estimated best-fit protein models for multi-gene real world datasets of the three domains of life. None of the available models is universally preferred for all alignments. *Source: [39]*

The same strategy was followed by Keane *et al.* in Modelgenerator (`http://bioinf.nuim.ie/modelgenerator/`) to obtain the results presented in [39]. Keane *et al.* conducted experiments on real world data-sets of the three domains of life. They found that for multi-gene phylogenetic inferences, there does not exist a model that universally fits for all alignments. The results are summarized in Figure 3.1. For bacteria, WAG was supported 46% of the time, whereas for eukaryotes, JTT was recommended most of the time (57%). For archaea, WAG and rtRev were supported in almost equal proportions (29% and 33% respectively).

Keane *et al.* concatenated alignments for different genes and computed the BIC and AIC as if it were one single gene. Pupko *et al.* proposed two additional methods to combine multiple genes in a single analysis in [59], the *separate model* and the *proportional model*. Figure 3.2 depicts the three concepts. They differ in the way branches of distinct genes are treated. The concatenated concept assumes the same branching history for all genes. The proportional model allows for different branch lengths, but assumes linkage in terms of a branch length ratio. Branch lengths can be completely different among genes in the separate model. Pupko *et al.* concisely describe the biological motivation by distinguishing the factors influencing the rates for each branch. Accordingly, for the separate branch concept, substitution rates are influenced by gene-driven factors as well as lineage-driven factors. The proportional model assumes the same influence from the lineage, while genes may evolve differently fast. For the concatenated concept, both factors contribute equally in every partition. According to Stamatakis *et al.* (see [73]), the concatenated and separate methods are also called *joint* and *per-partition* branch length optimization, respectively.

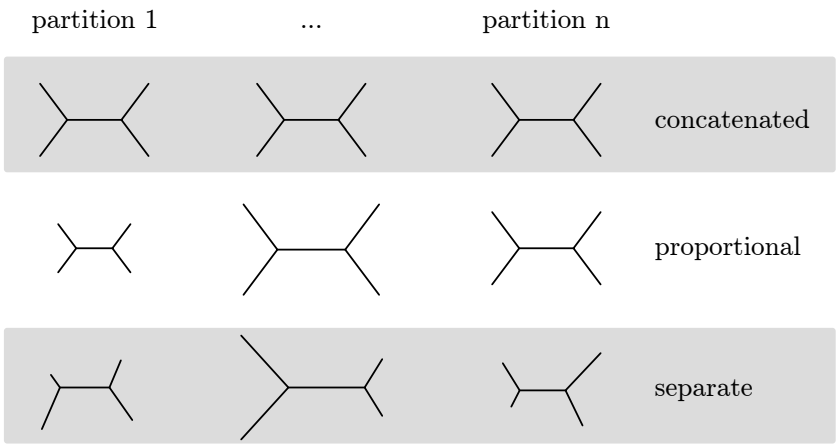


Figure 3.2: Methods to treat branch lengths for multi-gene phylogenies, as proposed by Pupko *et al.*.

Tanabe implemented the concatenated, proportional, and separate branching models in a multi-gene protein model selection application called Aminosan [76]. Aminosan also computes the AIC and BIC for concatenated, proportional, and separate branches. Whereas Aminosan is able to determine mixed model assignments for per-partition branch lengths, for joint-branches one common model is chosen for all partitions.

One model can be appropriate for homogeneous partitions (genes with similar substitution rates). However, this assumption may oversimplify the challenge for heterogeneous partitions. Recently Li and Rodrigo [45] assessed the covariation of branch

lengths and physically interacting genes. They found that the branches among genes correlate in these cases. That is, it is reasonable to assume linked branches between genes. Since Li and Rodrigo's findings also hold for only functionally related (physically non-interacting) genes, in practice, it might be important to assign different models for joint-branch ML inference.

3.2 Protein Model Assignment

In this thesis we focus on finding an adequate assignment of empirical protein models to multi-gene data, for ML tree inference with joint branches. That is, partitions are "linked via time", while the substitution rates may be different for each partition. We call this task *protein model assignment* (PMA). So far this issue has not been addressed.

In analogy to the supermatrix approach (Section 2.1), the biological meaning of PMA is that genes are assumed to evolve independently, whereas organisms have one single phylogeny. That is, we allow for gene-specific substitution rates while assuming a common branching history of genomes. The motivation is that genes do not have to evolve homogeneously (e.g., because some genes are more likely to become disadvantageous and thus thrown away by natural selection, than others).

Here we focus on the empirical protein evolution models listed in Table 2.3. As mentioned in Section 3.1, the most variation in the number of free parameters occurs in the rate matrix. For empirical protein models, the rate matrix does not include any free parameters. Furthermore, we compare model variations with the same options only, hence the number of free parameters is constant for all candidate models. Therefore, using AIC or BIC is pointless. Consequently, we compare different model assignments based on their log likelihood scores.

One can imagine the setup of PMA similar to cracking a combination lock. The number of partitions corresponds to the key size n. The set of candidate models corresponds to the code's alphabet, with m being the alphabet size. By rotating the discs of the lock one tries to figure out the key, whereas we optimize the likelihood score. As the key's length increases, so does the complexity of a brute-force attack. Similar to the combination lock, for the protein model assignment there are m^n assignments (combinations of m models to n partitions). The differences are that:

1. We cannot recognize the optimal solution without exhaustive search, whereas the key is optimal as soon as the lock opens.
2. We can apply decreases and increases of likelihood scores to determine the next

assignment to test (i.e., to guide the search), whereas for the lock all non-optimal solutions have the same score.

Assume a partitioned protein data set as outlined in Figure 3.3. Ideally, the task of adequately assigning models to partitions is to select that model for each partition such that the likelihood of the phylogenetic tree is maximized. That is, for all possible tree topologies, branch lengths and model assignments, return the configuration maximizing the likelihood. For unrooted binary trees there are, $\prod_{i=3}^{t}(2i-5)$ possible topologies (with t the number of taxa). An ideal approach would be to test each of the m^n model combinations for every topology. Moreover, a joint optimization of the $2t-3$ branches is necessary for each tree. Unfortunately, this ideal scenario is computationally infeasible.

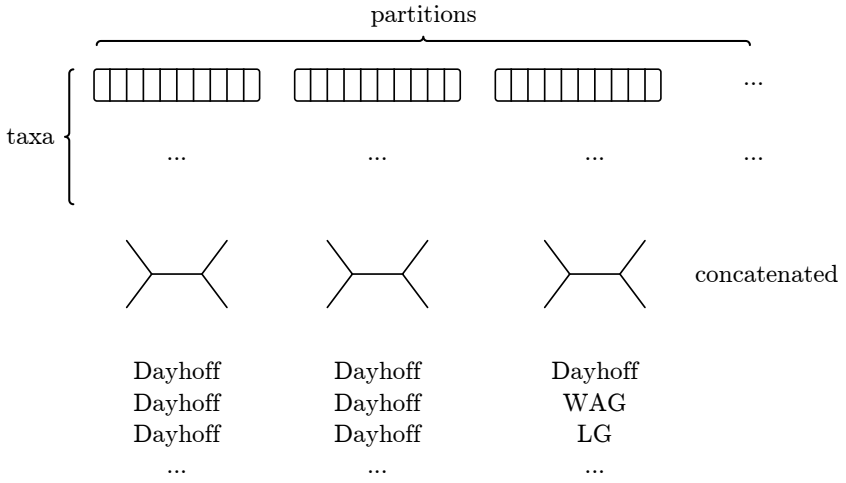


Figure 3.3: Concept and annotation of partitioned data. Tree topology is assumed to be fixed during model selection.

To reduce the complexity of the problem, a reasonable (i.e., non-random) fixed tree topology is assumed. Here we use a Maximum Parsimony tree for model assignment optimization. Of course, this can be a disadvantage. However, Posada and Crandall found [58] that for DNA evolution models, the tree topology does not influence model selection substantially, as long as it is reasonable. Therefore, using a parsimony tree for model assignment is an acceptable solution. We assume, that a "good" model assignment can be found on a fixed, reasonable tree. After all, models are simplifications of reality, only.

Note that models cannot be optimized independently on a per partition basis. This is due to the interdependent branch lengths. Optimal joint branch lengths may be different for each model assignment. Accordingly, a change of model m_1 partition 1 can decrease

the likelihood in the sense that the previously optimal models $m_{2...n}$ for partitions $2 \ldots n$ are not optimal any more.

In summary, we are maximizing the log likelihood lnL of the data for a fixed tree T with per-partition varying models D and jointly optimized branches e:

$$max(lnL(data|T, e, D))$$

Although we use a fixed tree topology, an exhaustive search for the best model assignment is still not feasible. The reason is the exponential large number of model combinations.

Chapter Summary

This chapter outlined the task of model selection. For mechanistic models, the number of free parameters in the substitution rate matrix varies greatly. Therefore many of these models are nested. Typically, hierarchical likelihood ration tests (hLRT) and the bayessian (BIC) or akaike (AIC) information criterion are applied to select an appropriate model. The goal is to balance the number of free parameters. Empirical models are mostly not nested. Therefore, usually only information criteria can be used for choosing empirical models.

Evolutionary properties may vary among different genes of the same organism. Therefore, in multi-gene DNA analyses model parameters are usually estimated on a per-gene basis. That is, substitution rates are allowed to be different for each partition. Optimal partitioned models are easy to determine for multi-gene analyses with per-partition branch lengths. However, it is difficult to assign models for protein analyses with joint branch length estimates. We call this the protein model assignment problem.

4 Algorithmic Approaches

Chapter 3 introduced the PMA problem. Because of the large number of potential assignments an exhaustive search is impractical for solving this task. Here we propose several heuristics for PMA. In particular, we present the adaptation of two well-known meta-search heuristics—Simulated Annealing and Genetic Algorithms.

Initially, we present a formal definition of the PMA problem and an overview of the search heuristics we have developed (Section 4.1). Thereafter, the algorithms for solving PMA are discussed in detail in Section 4.2.

4.1 Objectives and Overview

The protein model assignment problem is a "classic" combinatorial optimization problem. In analogy to the work of Blum and Roli [9], which offers a comprehensive overview of many search algorithms, we define the PMA problem $P = (S, l)$ as follows. Let

- $X = \{1, \ldots, n\}$ be a set of partitions
- D be the model domain (the set of models d)
- $l : \underbrace{D \times \cdots \times D}_{n\text{-times}} \rightarrow \mathbb{R}$ be the likelihood function that maps any model assignment to its corresponding likelihood score ($l(data|T, e, D)$)

The set of all candidate solutions (also *assignments*, or *configurations*) for the PMA problem is:

$$S = \{s = \{(1, d), \ldots, (n, d)\} | d \in D\}$$

That is, s assigns a model $d \in D$ to each partition i. Usually S is called the *search space*. The search space increases exponentially with the number of partitions n ($|S| = |D|^n$), where $|D|$ is the number of substitution models. The likelihood function l is the objective to be maximized. As already mentioned, the tree topology T is assumed to be fixed for the purpose of model assignment. Moreover, the data is fixed as well. Branches e must be optimized for every assignment, though. However, for each assignment, the

branch lengths are optimized using the same numerical optimization method (i.e., the Newton-Raphson method). Therefore, we define $l(s)$ to be the log likelihood score of assignment s with respect to T and optimized branch lengths e, given the observed data. That is:

$$l(s) = L_{data,T,e}(s)$$

In PMA we are striving to find an assignment $s^\star \in S$, which achieves a higher likelihood score than all other assignments $s \in S$:

$$l(s^\star) > l(s) \forall s \in S \setminus s^\star$$

$$\Leftrightarrow l(s^\star) = max(l(s))$$

We call $s^\star$ a globally optimal solution.

PMA is similar to many problems that are known to be NP-hard. At present it seems that PMA may be NP-hard as well. Thus, we suspect that there does not exist a deterministic algorithm that is guaranteed to find an exact solution in polynomial time, unless P=NP.

Here we focus on applying heuristic strategies to find sufficiently good assignments. Based on the basic properties of the algorithms we distinguish between *constructive* and *improvement* heuristics. Constructive algorithms generate an enhanced assignment from scratch, whereas improvement strategies aim at improving upon an initial solution of inferior quality.

Constructive Heuristics start with an initially empty solution. They successively add partition models to this initial assignment until a complete model assignment is constructed. The last step of these approaches includes the evaluation of the log likelihood of complete assignments under a joint-branch length estimate.

The PMA algorithms of this group deterministically compute a solution. That is, they always converge to the same result. Deterministic constructive heuristics are usually faster than improvement heuristics. They often return solutions of inferior quality, though.

Improvement Heuristics evaluate a set of initial, (mostly sub-optimal) random assignments. Starting from the initial assignments they successively apply changes to obtain better configurations. In contrast to the deterministic strategies, improvement heuristics evaluate all intermediate assignments (assignments in-between start and end

of the algorithm) completely and exactly.

Usually, improvement heuristics use some sort of randomness to more thoroughly explore the search space. Hence, they can converge to different assignments each time they are run. Moreover, if given infinite time, some improvement heuristics find the exact solution. These heuristics are often also called *meta search heuristic*.

4.2 PMA Heuristics

There exist multiple heuristic strategies for combinatorial optimization problems, which can be easily adapted to PMA. It is unclear which method is most appropriate. In this work fe focus on evaluating the most frequently used heuristic strategies. We assess two constructive and three improvement algorithms.

Initially, we present the constructive strategies—a naïve strategy based on per-partition likelihood scores in Section 4.2.1, and a greedy approach in Section 4.2.2. Thereafter, we discuss the more generic improvement heuristics for the PMA problem—a Hill-Climbing approach in Section 4.2.3, Simulated Annealing in Section 4.2.4, and a Genetic Algorithm in Section 4.2.5.

4.2.1 Naïve Heuristics

Changing the model of one partition influences the remaining partitions (via the joint branch lengths). Thus, the log likelihood score must be re-computed and branch lengths re-optimized for the entire dataset. Therefore, evaluating the score of a single model assignment is quite expensive. When branch lengths are optimized independently for every partition, the model assignment for one partition is independent from the remaining partitions. In this case we can construct the optimal assignment by just computing the optimal model for each partition infividually. Therefore, every model needs to be evaluated only once for each partition. The result is a model assignment s (however, with branches optimized per-partition). We call the best-scoring assignment under per-partition branches the *naïve optimum*. Accordingly, the per-partition model assignment optimization is referred to as the *naïve heuristics*.

To retrieve the naïve optimum, we compute a table r of size $|n| \times |D|$ (see Algorithm 1). The columns $i \in \{1, \ldots, n\}$ contain the likelihood score for partition i using model $d \in D$ in row d. Thereafter, the best naïve assignment s_{naive} can be constructed by extracting

and concatenating the optimal models on a per-partition basis. That is:

$$s_{naive} = \{(i, d|max(\{r_{i,d}\}))\forall i \in \{1, \ldots, n\}\}$$

Algorithm 1 Naïve Heuristics

```
1: naive ← table(n, |D|)          ▷ Initialize table of size n × |D|
2: for i ∈ {1, ..., n} do         ▷ Iterate over partitions
3:     for d ∈ D do               ▷ Iterate over models
4:         r_ij ← l({(i, d)})     ▷ Compute and save likelihood score
5:     end for
6: end for
```

The execution time of Algorithm 1 largely depends on the time needed for likelihood computations (line 4). Usually, computing the likelihood becomes faster as the alignment size decreases (w.r.t. the number of species and sites included). However, the likelihood computation includes numerical approximation operations to optimize branch lengths and the α-parameter of the Γ distribution of rate heterogeneity. As, these operations influence the computation of the likelihood, execution time may be different for distinct models. For the sake of simplicity, we ignore these factors here. The number of species is constant throughout the naïve heuristics, as are partition lengths. Therefore, we approximate the worst-case execution time for any likelihood computation by t_{max}. An upper bound for the execution time of Algorithm 1 (t_{naive}) can then be given by:

$$t_{naive} \in \mathcal{O}(|X| \cdot |D| \cdot t_{max})$$

The naïve optimum, can be retrieved afterwards by determining the maximum for each of the n columns of table r. Because usually r is comparatively small the naïve optimum can be obtained quickly.

Note that the naïve optimum does not need to be the optimum for PMA. To assess the difference between naïvely assigned models and optimal PMA, we calculated both for small alignments (see Section 6.3). For this purpose, we randomly extracted subsets from real data and computed the PMA solution exhaustively to compare it to the naïve optimum. On average the resulting model assignments differed in about 50% of the cases.

4.2.2 Greedy Assignment Composition

Usually, for large search-spaces it is tempting to assess greedy heuristics first. Algorithms are called greedy if they intend to find the solution of a problem by applying local improvements. The most common example for greedy algorithms is based on the change-making problem. Before introducing the greedy strategy for PMA we will briefly outline the change-making problem to illustrate the basic idea of such approaches.

Consider a waitress, that has to give change frequently. She wants to be efficient and always return as few coins as possible. The question is how to determine which coins she should return (i.e., the smallest number of coins to yield the change). The greedy method to select the coins consists of always choosing the coin with the highest value equal or smaller to the remaining change. For example, in the US for an amount of 48¢, return $25¢ + 10¢ + 10¢ + 1¢ + 1¢ + 1¢$. Luckily, in the US and many other countries the greedy algorithm yields the optimal solution (see [70]). However, the algorithm would not necessarily find the optimal solution, if some currency had coins of 1, 3, and 4. For example, assume a change of 6. For this problem instance, the greedy approach would return $4 + 1 + 1$, whereas the optimal solution obviously is $3 + 3$. This is a problem of greedy approaches, they do find a solution, but frequently that solution is sub-optimal.

Algorithm 2 Greedy Assignment Composition

```
1:  s ← ∅                                          ▷ Initialize an empty assignment
2:  for i ∈ {1, ..., n} do                         ▷ Iterate over partitions
3:      P ← ∅
4:      LH ← −∞
5:      for d ∈ D do                               ▷ Iterate over models
6:          if l(s ∪ (i, d)) ≥ LH then             ▷ Test model d for partition i
7:              P ← (i, d)
8:              LH ← l(s ∪ (i, d))
9:          end if
10:     end for
11:     s ← s ∪ P          ▷ Add optimal model for this partition to the solution
12: end for
13: return s                                       ▷ s contains the solution
```

Our greedy approach to PMA constructs a solution on a per-partition basis. Algorithm 2 formally describes this greedy strategy. Initially an empty solution s is generated. Thereafter, the optimal model assignment of the current partition i is computed (given the formerly optimized partitions $(j, d) \in s$, with $j < i$). That is, we start with an empty solution and iterate until the solution contains a complete assignment. Usually, longer partitions have a greater contribution to the likelihood score. Therefore, it can

be beneficial to initially sort X in descending order according to the partition lengths.

As for the naïve heuristics, the execution time of Algorithm 2 depends on the likelihood computations (line 6). Note that we cache the likelihood of line 6 to reuse it in line 8. For Algorithm 2 (in contrast to the naïve heuristics), the alignment length increases via the concatenation of distinct partitions (through the outer for-loop starting in line 2). Hence, we expect the execution time for the computation of likelihood scores to increase with each iteration of the outer loop. Consequently, we define t_j to be the execution time for the likelihood computation of an alignment containing partitions 1 to j, where $t_j < t_i \Leftrightarrow j < i$. Finally, we can approximate an upper bound for the execution time of Algorithm 2 (t_{greedy}) as follows:

$$t_{greedy} \in \mathcal{O}(|D| \cdot \sum_{i=1}^{n} t_i) = \mathcal{O}(|D| \cdot \frac{n(n+1)}{2} \cdot t_n)$$

$$\Leftrightarrow t_{greedy} \in \mathcal{O}(|D| \cdot n^2 \cdot t_n)$$

With n the number of partitions and $|D|$ the number of candidate models. However, the execution time for Algorithm 2 also depends on the distribution of the partition lengths. If partition lengths are constant, we may expect t_i to increase linearly with a constant $c \in \mathbb{R}$, (i.e., $t_i = c \cdot t_1 \cdot i$). For this case, we can narrow down the upper bound for the execution time of Algorithm 2 to:

$$t_{greedy} \in \mathcal{O}(|D| \cdot c \cdot t_1 \cdot \frac{n(n+1)}{2})$$

$$\Leftrightarrow t_{greedy} \in \mathcal{O}(|D| \cdot c \cdot t_1 \cdot n^2)$$

Analogously, if partitions at the beginning of the alignment are longer than those at the end, the algorithm is slower.

4.2.3 Hill-Climbing

Usually, Hill-Climbing starts with a random initial assignment (the *current best solution*). Thereafter, the algorithm repeatedly evaluates assignments in the *neighborhood* of the currently best assignment. For now, we define the neighborhood of an assignment s by:

$$N(s) = \{s' \in S | d_i = d'_i \wedge d_j \neq d'_j \forall j \in \{1, \ldots, n\}, i \in \{1, \ldots, n\} \setminus j\}$$

with d_i representing the model of partition i. That is, the neighborhood is the set of assignments that differ in only one model d_j. Note that the neighborhood is symmetric,

that is, $s' \in N(s) \Leftrightarrow s \in N(s')$.

If a better assignment can be found in the neighborhood, the procedure moves to the better assignment, that is, "climbs up-hill". In other words, the neighboring configuration is taken as new, currently best, assignment. Hill-Climbing implementations vary in the way they evaluate the neighborhood of an assignment. They can either move to better solutions immediately (*next ascent Hill-Climbing*), or evaluate the entire neighborhood and move to the best neighbor (*steepest ascent Hill-Climbing*).

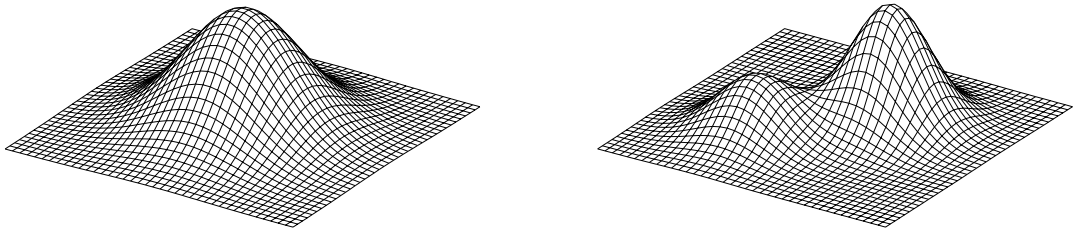
Hill-Climbing evaluates neighborhoods until it converges to a *peak*. That is, it converges if there is no better neighboring assignment for the currently best assignment. Such an assignment $\hat{s}$, with $l(\hat{s}) \geq l(s) \forall s \in N(\hat{s})$, is called *locally optimal* with respect to neighborhood N.

The Hill-Climbing approach to PMA is formally described by Algorithm 3. Initially, a random assignment is generated (line 1). For the currently best assignment, all neighboring assignments are generated and evaluated (line 5). When a better assignment is found, it is accepted (lines 6+7). Afterwards the neighborhood of the better assignment is searched. The procedure converges as soon as a local optimum is found (lines 11+12). In other words, Hill-Climbing is guaranteed to find a local optimum.

Algorithm 3 PMA by Hill-Climbing

```
1:  s ← random ∈ S, l_max ← l(s)                ▷ Initialize random starting solution
2:  while true do
3:      for x'_i ∈ X do                          ▷ Iterate over partitions
4:          for v'_i ∈ D do                      ▷ Iterate over model
5:              s' ← {(x_j, v_j) ∈ s | j ∈ {0, ..., n} \ i} ∪ (x'_i, v'_i)
6:              if l(s') > l(s) then             ▷ Evaluate neighbor model
7:                  s ← s'                       ▷ Accept neighbor
8:              end if
9:          end for
10:     end for
11:     if l_max > l(s) then                     ▷ Found no better neighbor
12:         return s                             ▷ s must be a local optimum
13:     end if
14:     l_max ← l(s)                             ▷ Move to best neighbor
15: end while
```

For some problem instances, there may not exist multiple local optima (Figure 4.1(a)), hence a local optimum is also the global optimum. In contrast, Figure 4.1(b) illustrates a search-space with two local optima. When using a Hill-Climbing algorithm, only the lower peak may be encountered. This is the major disadvantage of Hill-Climbing. Figure 4.2 depicts a search-landscape with multiple local optima (black points). Which



(a) One local optimum (b) Two local optima

Figure 4.1: Search-landscape illustration for a hypothetic two-dimensional maximization problem ((a) has one local optimum, whereas (b) shows an objective function with two local optima).

peak will be found depends on the position of the initial assignment. Assume that point b is the global optimum. If one starts with a solution close to point a (in the surrounding borders of a) the Hill-Climbing method will not converge to the global optimum. An initial solution within the borders of b, however, would return b.

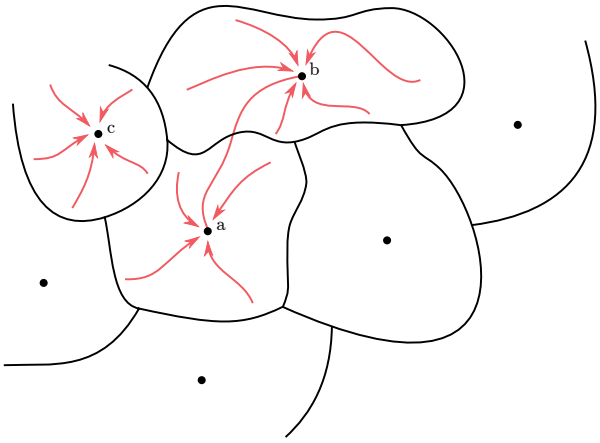


Figure 4.2: Plan view of a hypothetic search-landscape with multiple local optima (points a to g). b is the global optimum. Local search will not find b, if the search is initialized with configuration near to an other local optimum. *Adapted from* `http://en.wikipedia.org/wiki/Local_optimum`, *4/16/2012*

The number of local optima depends on the definition and size of the neighborhood of an assignment. Therefore, also the probability of finding an only locally optimal assignment depends on the neighborhood. Because we do not know better, we suspect that the PMA search-space contains multiple local optima, unless we define all assignments to be neighbors. Hence, Hill-Climbing will probably converge to a local optimum. Note, however, that the neighborhood-size must be chosen carefully. That is, there exists a trade-off between speed and accuracy. If the neighborhood is too large, the algorithm becomes inefficient. If the neighborhood is too small, the algorithm can easily and

frequently get stuck in a local optimum.

Our neighborhood definition from above assumes that any model may be the optimal choice for any partition. That is, to get a neighbor any partition may be assigned another random model (we do not prefer a model or partition a priori). Consequently, there are $(|D| - 1) \cdot n$ neighbors. Given that we do not want to bias our choice toward some models or partitions, there does not exist a smaller neighborhood.

To decrease the probability of finding only locally optimal assignments, Hill-climbing can be rerun with different initial solutions. Furthermore, the neighborhood may be successively increased in every run. This approach is also known as *variable neighborhood search*. For this purpose, we specify $N_k(s)$, with a constant $k \geq 1$ that determines the neighborhood of assignment s in run k. As we do not want to specify the set of partitions that vary nor the models, the neighborhood can only be increased by assigning different models to more partitions. In other words, for $k = 1$ one model may be different, for $k = 2$ two models can be changed, and so on. Therefore, as k increases the neighborhood size increases, as well. The size of N_k can be given as follows:

$$|N_k| = \underbrace{\binom{n}{k}}_{\text{partition subsets}} \cdot \underbrace{(|D| - 1)^k}_{\text{model combinations}}$$

For a small data-set consisting of four partitions, a variable neighborhood search with $1 \leq k \leq 3$ would at least evaluate 21,454 ($N_1 + N_2 + N_3 = 4 \cdot 17 + 6 \cdot 17^2 + 4 \cdot 17^3$) neighbors (out of 104,976 possible assignments, or approx. 20% of the search-space). However, we consider the evaluation of only one neighborhood here (remember that if better neighbors are found the search continues with either the best neighbor or the first better neighbor). That is, for small alignments the variable neighborhood search may degenerate to an almost exhaustive search. For large alignments, this danger decreases. However, this is mainly due to the rapid increase of the search space (e.g., for $n = 50$ and $1 \leq k \leq 3$ the minimal number of neighbors to evaluate represents much less than 1% of the search-space). Because of the large number of likelihood evaluations we did not exploit the variable neighborhood search for PMA.

4.2.4 Simulated Annealing

The Simulated Annealing (SA) technique is inspired by the annealing process in thermodynamics. Its thermodynamic purpose is to increase the stability of metals. During this process, the metal is initially heated and cooled slowly afterwards to reach a low-energy

state. The lower the energy, the more stable the metal. The approach was independently proposed by Kirkpatrick *et al.* in [42] and Černỳ in [14]. Both papers apply the method to the well-known NP-hard traveling salesman problem.

Compared to Hill-Climbing, the advantage is that, SA always accepts better states, but also allows for moves that generate assignments of worse quality. Worse moves are often called *backward steps* (for maximization problems also *downhill steps*). Accepting worse moves allows the search to move out of local optima and thereby more thoroughly explore the search-space. Usually, the probability of backward steps decreases during the SA search. Because SA can escape from local optima it is frequently applied to NP-hard optimization problems. Moreover, it can be easily adapted to the specific problem at hand.

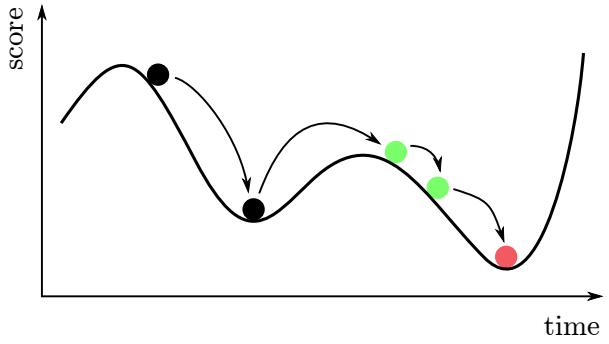


Figure 4.3: Concept of Simulated Annealing. *Adapted from* `http://www.iasor.tu-clausthal.de/Arbeitsgruppen/Stochastische-Optimierung/forschung/sa`, *4/16/2012*

Often, optimization problems are formulated as minimization problems. The PMA problem is a maximization problem. This can easily be formulated as minimization problem by multiplying log likelihood scores by -1. We outline the basic SA procedure for minimization problems, because this allows for constructing an intuitive example. Assume we want to minimize the objective function depicted in Figure 4.3. Let the leftmost ball be an initial random solution. In SA, the ball will always roll down during the search. If the ball is fast (high temperature), it will probably also be able to traverse a small hill (slightly worse solution). For example, the first hill in Figure 4.3 is traversed (the red moves). However, as the ball becomes slower (as the temperature decreases) it becomes more unlikely to climb another hill (accept backward steps). Also, it is less likely, that a large hill will be traversed (a considerably worse solution is accepted). For example, in Figure 4.3 there is not enough speed left to climb the second hill. The green ball depicts the final solution returned by the algorithm.

An adaption of SA to the PMA problem applies this principle to maximize the likeli-

hood. Again, we start with a random assignment s and its likelihood score $l(s)$ (lines 1 to 2 of Algorithm 4). Line 3 initializes the counter k that determines the temperature. Thereafter (lines 5 to 14), the algorithm successively evaluates the neighbors $s' \in N(s)$ of s (thus, as for Hill-Climbing, the neighborhood should be chosen with care). If a neighbor s' with $l(s') > l(s)$ is found, s' is accepted immediately (lines 6 to 8). That is, the remainder of the neighborhood will not be evaluated. If $l(s') \leq l(s)$, the algorithm moves to s' with a certain *acceptance probability* $p(s, s', T) \in [0; 1]$. The acceptance probability is a function of the temperature T and the likelihood scores of s and s'. T decreases monotonically during the search process, and so does the acceptance probability. That is, in the beginning, the procedure is more likely to accept a worse move than in the end. Furthermore, every point in time a small score decrease is accepted with higher probability than a larger one. Usually, the acceptance probability is defined as $p(s, s', T) = e^{-\frac{l(s)-l(s')}{T}}$, also known as *Metropolis criterion* (see [1]).

Algorithm 4 Simulated Annealing Algorithm for PMA

```
1:  s ← random ∈ S                                   ▷ Initialization
2:  s_max ← s, l_max ← l(s)
3:  k ← 0
4:  repeat
5:      s' ← next ∈ N(s)                             ▷ Get next neighbor
6:      if random ∈ [0;1] < e^(-(l(s')-l(s))/T) then
7:          s ← s'                                   ▷ Move to worse neighbor
8:      end if
9:      if l(s) ≥ l_max then                         ▷ If new maximum
10:         s_max ← s                                ▷ replace old maximum
11:         l_max ← l(s)
12:     end if
13:     k ← k + 1                                    ▷ Prepare next temperature
14: until T_k > 0                                    ▷ As long as not cold
15: return s_max                                     ▷ s_max contains best solution
```

For each iteration k of the loop (lines 4 to 14), the temperature T_k it is given by an *annealing schedule*. The schedule consists of a starting temperature T_0 and a function $\tau : k \to \mathbb{N}$. There exists a large variety of distinct definitions for τ. Even a non-monotonically decreasing τ may be applied. Usually, the following cooling strategy is used:

$$\tau : T_k = \lfloor T_0 \beta^k \rfloor$$

with $\beta \in [0; 1]$. We evaluate different choices for T_0 and β in Section 6.3.2. SA converges when the system is totally cool ($T_k = 0$).

4.2.5 Genetic Algorithm

Genetic Algorithms (GAs) can also escape from local optima. A GA mimics the process of evolution. The terminology used in this context is derived from biology. We refer to *individuals* (assignments) and their *fitness* (likelihood), as well as *selection* and *reproduction* (moves). GAs regard solutions as genetic properties of individuals. A set of individuals is called a *population*. The motivation is that the individuals of a population include beneficial and disadvantageous "genetic" properties. Mating of individuals is thought to yield fitter (better adapted) individuals. Therefore, an improved assignment is created by evolving previous assignments. There exists a large variety of different implementations of GAs. Here we discuss our adaption of GAs to the PMA problem. In [33], Haupt *et al.* offer a detailed and comprehensive overview of implementation options for GAs.

Typically, GAs start with a random population of individuals (the current population). Thereafter, some individuals of the population are chosen to produce new individuals to replace the current population. We call the population $g_t \subset S$ of a given iteration $t \in \mathbb{N}$ of the search, the t^{th} generation. Frequently, all generations have the same size N_{pop}, the *population size*. Except of g_1 (the initial population), new generations g_{t+1} are created from individuals of the current generation g_t. Accordingly, g_{t+1} is called the *offspring* of g_t.

The fitter $s \in g_t$, the higher its chance to reproduce. However, similar to nature, every individual should have the chance to reproduce, in order to explore the search-space more thoroughly. Moreover, to increase diversity, *mutations* can be applied during the reproduction phase.

There does not exist a strict, commonly accepted definition for GAs. Usually the following three components are present:

1. A *selection strategy* is applied to choose the parents of generation g_t. Here, we apply a *tournament* for parent selection. That is, for each parent we select two candidate assignments from the current population at random. The fitter candidate is chosen as the first parent s_1. To select the second parent s_2, the tournament is repeated. We refer to parent selection by $select(g_t)$, which returns exactly one parent.
2. A *(re)combination procedure* that exchanges parts of s_1 and s_2 to generate two new individuals s_{c1} and s_{c2}. That is, both children $s_{c1,2}$ consist of "genetic material" of the two parents. For this purpose, usually, the chromosomal crossover of evolution is emulated. We apply a single point crossover strategy. That is, s_1 and s_2 are

split at the same random position $i \in \{1, \ldots, n-1\}$, with n being the number of partitions. Afterwards, the first part of s_1 is combined with the second part of s_2 into s_{c1}. For s_{c2} the second part of s_1 and the first part of s_2 are combined. Formally, that is:

$$s_{c1} = \{\{(i, d_i) \in s_1\} \cup \{(i+1, d_{i+1}) \in s_2\}\}$$

$$s_{c2} = \{\{(i, d_i) \in s_2\} \cup \{(i+1, d_{i+1}) \in s_1\}\}$$

Figure 4.4 illustrates the single point crossover. We refer to single point crossover by $crossover(s_1, s_2)$, which returns the children s_{c1} and s_{c2}.

3. A *mutation strategy*, to increase diversity of populations. Usually each gene of an assignment s of the offspring generation is changed with a small probability—the *mutation probability*. We mutate every partitions assignment into another random model with probability P_m. This operation is denoted as $mutate(s)$, which returns the mutated assignment s'.

   Increasing diversity by mutation is especially important because all individuals of the initial population could coincidentally share one or more properties (same model for a partition for all assignments). Applying the crossover technique these properties would never change.

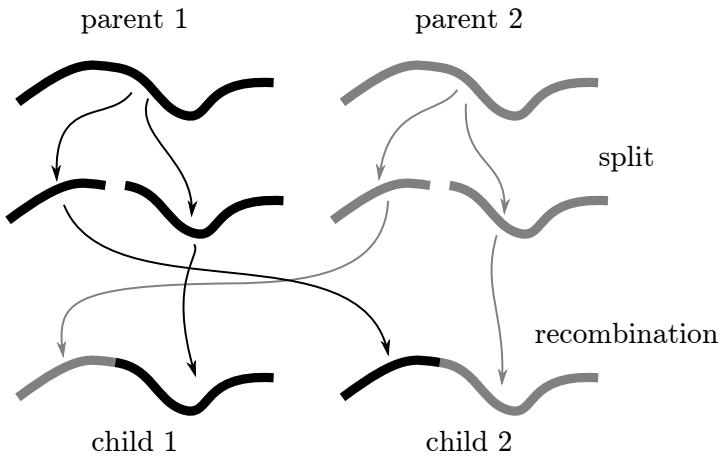


Figure 4.4: Schema of the single point crossover procedure.

Until now we did not discuss convergence of the GA approach. Obviously, the algorithm could simply converge after a predefined number of populations has been evaluated. We want the search to converge, if it does not noticeably improve the search-result. Therefore, the algorithm terminates as soon as an offspring did not contain a better individual.

Algorithm 5 Genetic Algorithm for PMA

```
1:  t ← 0, g(t) ← {random ∈ S}                          ▷ Initialization
2:  s* ← ∅
3:  while true do
4:      s* ← fittest(g_t)                    ▷ Determine best the currently best solution
5:      for 1, ..., N_pop do                                        ▷ Create offspring
6:          s_1, s_2 ← select(g_t)                          ▷ Select parents s_1 and s_2
7:          s_child ← crossover(s_1, s_2)                   ▷ Recombination of parents
8:          s_child ← mutate(s_child)                       ▷ Occasional mutation
9:          g_{t+1} ← g_{t+1} ∪ s_child
10:     end for
11:     if l(s*) > l(fittest(g_{t+1})) then
12:         return s*                                       ▷ No further enhancement
13:     end if
14:     t ← t + 1
15: end while
```

Algorithm 5 outlines the GA procedure for PMA. It starts by initializing a random set of assignments for the current generation g_1 in line 1. For simplicity, we define the procedure $fittest(g_t)$, which returns the best assignment of generation g_t. In line 3 we determine the currently best solution. Thereafter, N_{pop} individuals are selected according to the operations in lines 4 to 9. If a better assignment was found in the current population the search continues, otherwise the currently best assignment is returned in line 12.

Chapter Summary

In this chapter we formally introduced the protein model assignment (PMA) problem. The PMA problem is a classic combinatorial optimization problem. The goal is to maximize the log likelihood score of a fixed reasonable tree (i.e., a parsimony tree) for given data, by varying partitions model assignments. That is, given the data and the tree, we want to find a model for each partition, so that the overall likelihood is maximized. As we want to jointly estimate the branch length any combination of models $d \in D$ to partitions $i \in \{1, \ldots, n\}$ may be optimal, hence, an exhaustive search does not see feasible for many partitions.

Therefore, we developed differentiated heuristics that help to efficiently search for "good enough" model assignments. In particular we introduced two deterministic approaches (the naïve heuristics, and the greedy assignment composition) and presented the adaption of three non-deterministic heuristics (Hill-Climbing, Simulated Annealing,

and Genetic Algorithm) to PMA. Both deterministic strategies create a solution from an initially empty assignment. Therefore, they are also referred to as *constructive heuristics*. The non-deterministic approaches, in contrast, guess a random assignment, and aim at improving upon its score by repeatedly applying small changes. Therefore, these heuristics are frequently called *improvement heuristics*.

5 Improving Performance

Note that the improvement heuristics of Section 4.2 are generic algorithms and do not employ problem-specific knowledge to solve the problem (they only compute likelihood scores). Thus, such methods are often denoted as *black-box* searches. In [81], Wolpert and Macready found that search algorithms preform according to the quality and amount of domain-specific knowledge that is included. Here, we develop adaptations (often also called *hybrid heuristics*) to reduce execution times or improve the accuracy of the heuristics.

Section 5.1 outlines various combinations of heuristic approaches. Thereafter, we describe techniques that strive to reduce the number of candidate models (Section 5.2). In Section 5.3 we discuss approaches to speed up likelihood computations.

5.1 Seeding and Pipelining

Essentially, Hill-Climbing and SA differ in the way by which they move to and accept new solutions. Remember that, SA is able to accept worse solutions (downhill steps) with some probability. However, the basic underlying principle is similar to Hill-Climbing. Furthermore, SA can be considered to be a GA, with a population size of one. Because the size of all generations is one, crossover operations are not considered. The mutation operation can be defined to transfer assignments into a neighboring configuration with probability 1.0. However, usually the population size is much larger. Hence random improvement heuristics are often subdivided into *single-solution* and *multiple-solution* based heuristics. SA and Hill-Climbing are single-solution based, whereas GAs are multiple-solution based. This is also the reason why SA and Hill-Climbing appear to be more similar than SA and GA.

The major difference between improvement heuristics lies in the implementation of *intensification* and *diversification*. That is, the exploration of high quality areas of the search-space and the exploration of yet unvisited areas, respectively (see [9, 86]). Usually, a balance between these two criteria can be achieved by adjusting heuristic parameters.

For example, decreasing the temperature of SA, will favor intensification. A similar effect, although via a distinct mechanism, can be achieved by decreasing the population size in GAs. Usually, GAs exhibit a higher proportion of diversification, because they are initialized with several random assignments. Hill-Climbing only deploys intensification.

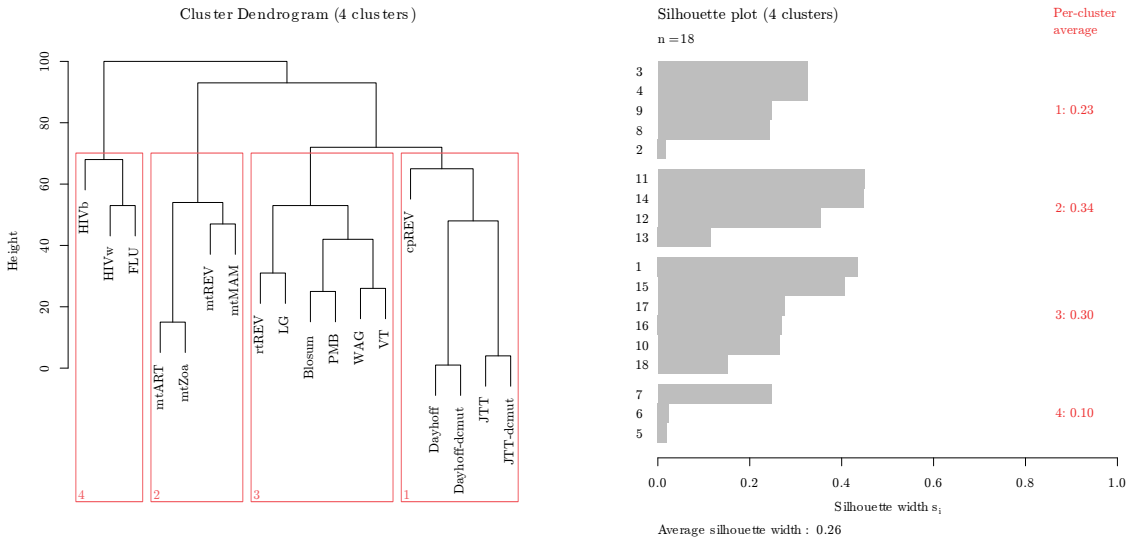
Besides parameter optimization combinations of the basic heuristics are frequently exploited to improve performance. That is, the self-contained heuristics are executed in sequence (one after the other). In other words, results of earlier algorithms are passed to later algorithms. Thereby, benefits of each algorithm can be leveraged, while overcoming its weaknesses. For example, the deterministic constructive heuristics of Section 4.2 have the benefit of predictable (usually relatively short) execution times. Moreover, they frequently find "good" (on average better than random) solutions. However, these heuristics may return suboptimal results. On the other hand, Hill-Climbing seems to be well-suited for improving upon a specific solution and thereby finding a better assignment in a small part of the search-space. However, because Hill-Climbing is initialized by a random assignment, it may spend a lot of time on improving a low quality initial configuration to obtain a "good" assignment. A straight-forward enhancement is to initialize Hill-Climbing with a "good" solution. In other words to *seed* the heuristic with a good initial configuration. Deterministic heuristics can be applied for this purpose. That is, the result of greedy assignment composition or the naïve heuristics, serve as initial solution to the Hill-Climbing algorithm. More generally, also SA and GA may be seeded with results of these algorithms.

Furthermore, one could execute any subset of random heuristics one after the other in a *pipeline*, always propagating the results to subsequent pipeline stages. For instance, it is generally accepted that the strength of GAs is to efficiently locate regions of good quality [16]. Due to the comparatively high amount of diversification operations, GAs quickly leave these regions again without thoroughly exploring them. Therefore, it may be promising to apply GAs to create seed populations of "good" quality and refine these assignments by Hill-Climbing afterwards. However, such a pipeline may increases the overall execution time dramatically. Therefore, we only exploit seeding in this thesis.

5.2 Search-Space Reduction by Model Clustering

As already mentioned in Section 3.1, biologists sometimes assign models to partitions based on empirical criteria. The approach presented here is somewhat similar. By computing similarities between models, we intend to reduce the number of candidate models, that is, reduce the size of $|D|$ in $|D|^n$ (the number of posible model assignments). For

this purpose, the most similar models (models that specify similar substitution rates and that are hence thought to be likely to yield similar results in terms of log likelihood scores) are clustered. Thereafter, a representative average model is calculated for each cluster. Instead of evaluating every individual model, these representative/cluster-models serve as placeholders for an initial model assignment. That is, the cluster-models are evaluated first. Then, we focus on evaluating the individual models in the corresponding clusters for every partition. In other words, we then further refine the model assignment. Therefore, this approach separates the PMA problem into two problems. Firstly, we strive to solve a PMA instance with a different (i.e., smaller) set of candidate models. This task assigns a cluster of substitution models to each of the partitions. Subsequently, we replace the set of candidate models D of the initial PMA instance by n distinct sets $D_i \subseteq D$, the sets of candidate models for partitions 1 to n (each D_i contains the models of the cluster that was assigned to partition i).



(a) Clusters (b) Silhouette

Figure 5.1: Hierarchical clustering based on the element-wise euclidean distance of mutation rates. Figure 5.1(a) depicts the cluster dendrogram. Figure 5.1(b) depicts the corresponding silhouette plot for four clusters.

To compare the *relative* rates of different substitution matrices, matrix-entries need to be scaled. We do this by dividing each matrix-element by the maximum rate of the matrix ($\frac{q_{ij}}{max\{(Q)\}}$, with $i, j \in \{1 \dots 20\}$). Afterwards, the entries of the scaled matrices are used to calculate the euclidean distances for each pair of matrices. That is, for two matrices $\{a_{ij}\}$ and $\{b_{ij}\}$ the distance is $\sqrt{\sum_{i=1}^{20} \sum_{j=1}^{20} (a_{ij} - b_{ij})^2}$. Consequently, we calculate a symmetrical $|D| \times |D|$ distance matrix. Each entry of the distance matrix

describes the distance between models of the corresponding column and row (diagonal entries are 0.0). Based on these distances, complete-linkage hierarchical clustering (see [41]) can be applied to cluster substitution matrices.

Figure 5.1(a) depicts the resulting cluster-dendrogram for the models of Table 2.3. It can be observed that Dayhoff and Dayhoff-dcmut, as well as, JTT and JTT-dcmut—as may be expected—have similar rates. In fact, both model-pairs were created using the same data to estimate the substitution rates. However, the dcmut versions employ a slightly different mathematical method to compute substitution rates [43]. Moreover, moving from left to right in Figure 5.1(a) one can observe four groups of substitution models (i.e., retroviral models, mitochondrial models, general models, and models created with an approach similar to the one of Dayhoff and Schwartz [20]).

However, the clusters are not very dense and well separated. That is, the distances within clusters are not significantly smaller than those between clusters. Frequently, the *silhouette coefficient* (see [68]) is applied to objectively measure a given clustering is supported. For this purpose, we define the *silhouette* $s(a)$ of any object a (substitution rate matrices) in cluster A as the difference of a's average dissimilarity (distance) to the nearest next cluster B (the average of all distances $d(a, b)$ with $b \in B$) and its distance to A, divided by the maximum distance to any object a' in A or B. Therefore, the silhouette $s(a)$ is determined as follows:

$$s(a) = \frac{d(a, B) - d(a, A)}{max(d(a, b), d(a, a'))}$$

with $d(a, B) = \frac{1}{|B|} \sum_{b \in B} d(a, b)$, and $d(a, A)$ analogously. Then the silhouette coefficient S_c is the average of the silhouettes of all objects for a given clustering (note that $S_c \in [-1; 1]$). Values close to 1.0 reflect relatively smaller within cluster distances than between cluster distances (objects are well classified). A negative cluster coefficient, in contrast, implies that some objects are closer to the neighboring cluster than to their actual cluster. Figure 5.1(b) depicts the silhouettes for each substitution model w.r.t. the four clusters of Figure 5.1(a). Moreover the per-cluster average silhouettes and the silhouette coefficient are provided.

We computed S_c for all possible clusterings for the dendrogramm of Figure 5.1(a). Assigning 12 clusters results in the maximum silhouette coefficient of 0.38. Usually, for values $S_c \leq 0.5$ clusterings are thought to be supported only weakly by the data.

To assess if our clustering is useful, we must show that models within one cluster usually yield similar likelihood scores. For this purpose, we evaluated the likelihood scores for data sets of each of the three domains of life (see Section 6.1) with per-

partition branch length estimates using each model. Based on these values we compute corresponding partition distance matrices using the per-partition likelihood scores (one distance matrix per partition per data set). Thereafter, the partition distance matrices are merged into a comprehensive (data set distance) matrix, by averaging. If our clustering is useful, we expect the log likelihood scores to cluster in a similar way as the substitution rates. In other words, we would expect a dendrogram similar to that of Figure 5.1(a) by applying complete-linkage hierarchical clustering. However, we obtained quite distinct likelihood-based clusters for all data sets (e.g., see Figure 5.2). Even the closely related models (JTT and JTT-dcmut) produced relatively distinct likelihood scores. Consequently, the above clustering of substitution rates does not appear to be promising since it does not correlate well with log likelihood scores.

One explanation may be that we did not include information about the data at hand during model clustering (the clusters are solely based on substitution rates).

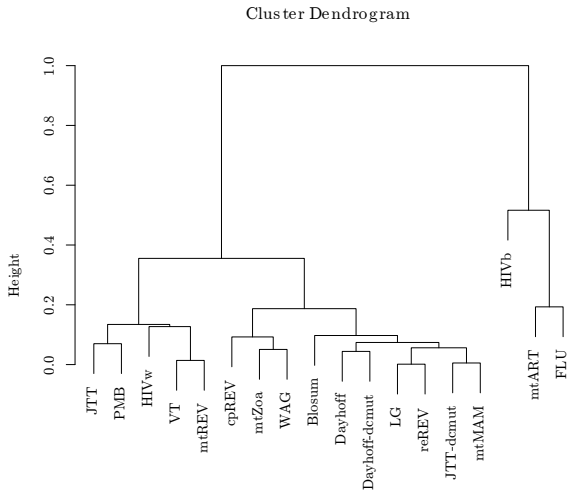


Figure 5.2: Hierarchical clustering over all partitions based on log likelihood score distances for mammalian dataset of [48].

To address this problem, we could introduce a data-aware log likelihood based clustering approach for each partition. That is, we can apply the clustering for each partition to obtain model-clusters that actually yield similar scores for a specific partition. This information could subsequently be applied as proposed in the beginning of this section. However, we did not exploit this approach, yet.

5.3 Reducing Assignment Evaluation Costs

The operation dominating run times in all PMA heuristics is the evaluation of the likelihood for different model assignments. In this section we introduce two strategies for reducing the computational demands.

Firstly, we focus on reducing the cost for recurring computations in Section 5.3.1. Secondly, to reduce the overall number of assignments that must be evaluated, we present a strategy to quickly pre-score assignments (Section 5.3.2).

5.3.1 Lazy Likelihood Computations

As mentioned in Section 4.1, every assignment is evaluated using exactly the same tree topology, and numerical optimization techniques for optimizing branch lengths and the α-shape parameter of the Γ distribution for among site rate variation. In fact, it is important to keep the topology fixed to be able to compare the likelihoods of different assignments, such as not to confound the tree search and model assignment steps. As, we usually only apply small changes to the protein model assignment (e.g., changing one model for a single partition at a time) during the search process, it is rather unlikely that the branches or α change drastically. Hence, we want to avoid completely re-optimizing these parameters. Consequently, instead of resetting branch lengths and α to their initial standard values, we invoke the optimization routines with the values of the previous assignment (i.e., we refine α and the branches). In our tests this modification only generated minimal deviation in log likelihood scores (smaller than 0.5 log likelihood units).

Furthermore, randomized search algorithms, sometimes re-inspect the same assignments. To avoid re-computing the likelihood of such assignments, we maintain a list that archives likelihood scores for assignments that have already been inspected. However, to avoid re-computing likelihoods, it is necessary to lookup each assignment before evaluation. Depending on the data-structure chosen to implement the archive, this task can become quite expensive. Also, as typically only few assignments are evaluated more than one time, it is not necessarily worth to maintain and look-up the archive. Moreover, it may not be worth to develop and implement elaborate data structures for the archive.

Because of the corresponding low development effort we implemented the archive as unsorted list for an initial evaluation. That is, assignments are appended to the end

of the list when they are evaluated. Thus, the list must be scanned sequentially before every assignment evaluation. Applying a hash table would certainly be superior w.r.t. the access time. However, our analyses in (see Section 6.3.4) revealed short archive look-up times. Therefore, we did not evaluate using a hash table for the archive, yet.

5.3.2 Approximate Assignment Scoring

Both approaches presented in Section 5.3.1 promise to reduce execution time for PMA heuristics by avoiding unnecessary work. Therefore, these approaches do not, or only slightly, influence the quality of results. Here we develop a technique that promises to reduce execution time even more. It may cause worse results, though. In analogy to the work of Rana *et al.* in [61], we aim at quickly identifying "low quality" assignments. That is, assignments that are not expected to improve upon a currently best assignment. We call the expected quality of an assignment its *potential*. Because likelihood computations dominate the execution time of PMA heuristics it would be beneficial to avoid evaluating assignments of low potential.

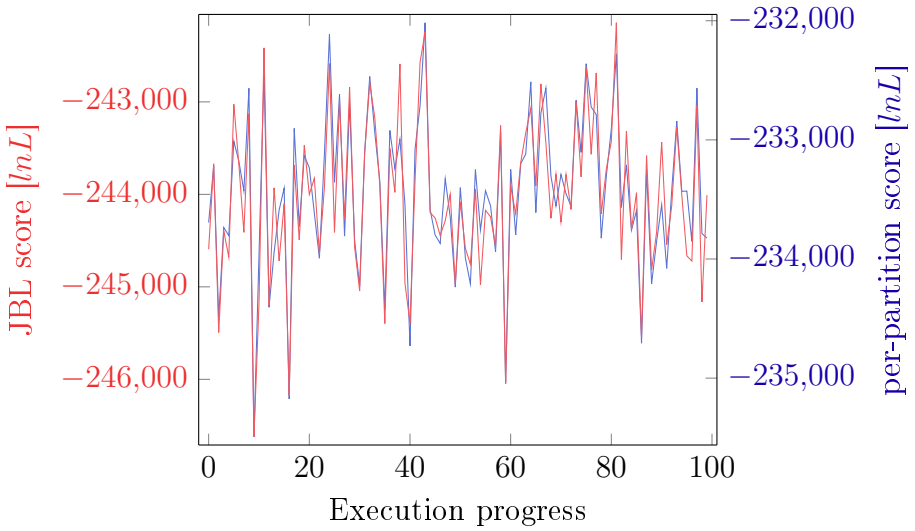


Figure 5.3: Log Likelihood scores of random 100 model assignments under joint and per-partition branch length estimates (correlation is 0.93).

In order to provide the rationale for the approach we develop in this section it is important to note that we observed a strong correlation between likelihood scores using per-partition $ibl(s)$ and joint $l(s)$ branch lengths estimation in many initial analyses. That is, a peak in the likelihood function for a joint branch length estimate frequently coincides with a peak of an independent branch length estimate. Figure 5.3 depicts

likelihood scores (joint and per-partition) for 100 random assignments of a 50 partitions and 50 taxa data sample (the sample was extracted from the eukaryotes data set, see Section 6.1).

It can be easily observed that the scores are correlated (the pearson correlation coefficient is 0.93). However, as already mentioned in Section 4.2.1, the optimum for per-partition branch lengths optimization may be different from the optimal PMA assignment. Because of the strong correlation, it seems to be reasonable to approximate the potential of an assignment from its per-partition likelihood score, though. In other words, we approximate the potential of an assignment before actually evaluating it. Only if the potential is promising, we compute the exact likelihood. For this purpose we define the potential of an assignment s by a function $p_S : s \rightarrow \{true, false\}$ (we evaluate s only, if $p_S(s) = true$). To determine $p_S(s)$ we include values of the likelihood score of per-partition branch length estimates that can be computed quickly. In particular, we exploit the following three statistic properties:

1. the mean $\overline{ibl(s \in S)}$ of all assignments in S
2. the accumulated average $\overline{ibl(s_j)}$ with s_j the assignments that have already been visited
3. the moving average $\overline{ibl(s_w)}$ of the w most recently visited assignments

for per-partition branch length estimates (ibl). Therefore, for example, p may be defined as follows:

$$p_S(s_i) = ibl(s) - \overline{ibl(s \in S)} > 0 \wedge ibl(s) - \overline{ibl(s_w)} > 0 \wedge ibl(s) - \overline{ibl(s_j)} > 0$$

to exploit all properties simultaneously.

Which measures are helpful may depend on the data. Figure 5.4 depicts the proposed statistics, as well as the JBL and per-partition likelihoods, for an execution of the Hill-Climbing heuristic on the sample of the eukaryotes data set from above. It can be observed that the mean of per-partition likelihood scores is comparatively low (most of the evaluated assignments have better per-partition log likelihood), and therefore might not very well suited for determining if an assignment is promising. Although the accumulated average increases as the likelihoods of the evaluated assignments increases, it is also underestimates what would be expected to be a promising assignment. In other words, if we only apply the mean and accumulated average, $p_S(s)$ would evaluate to true for most assignments. However, we expect these measure to be helpful to avoid the evaluation of many poor assignments in the initial phase of Hill-Climbing and SA.

We expect the moving average to be most suitable to approximate if an assignment is

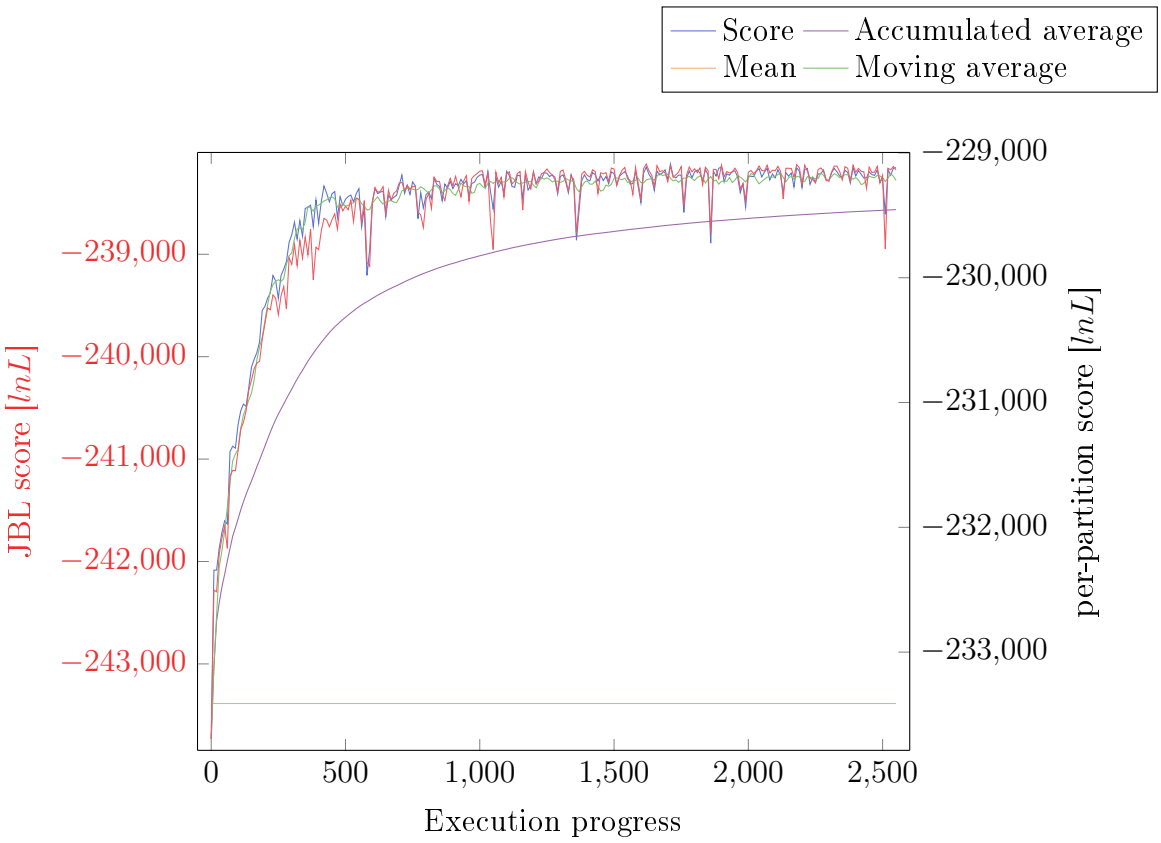


Figure 5.4: Hill-Climbing algorithm on the data sample of Figure 5.3 (due to readability only every 10^{th} data point is plotted). The plot includes the likelihood scores of the current assignment for JBL and per-partition branch lengths estimation. Furthermore, the mean, the moving average, and the cumulated average for per-partition branch lengths estimation are included.

promising. Moreover, we can adapt the window size to decrease or increase the number of evaluated assignments. If we apply a window of size 1, the moving average is identical to forcing a per-partition likelihood score better than that of the former assignment. Choosing very large window, on the other hand, brings the moving average closer to the accumulated average. We expect a relatively small window (e.g., $\{2, \ldots, 10\}$) to perform best. Thereby, we mean that it allows the evaluation of comparatively good assignments, always depending on the current "quality level" of the search. We present the evaluation of all criteria in Section 6.3.4.

Chapter Summary

In this chapter we developed several approaches to improve the performance of the PMA heuristics of Chapter 4.

Firstly, we discussed how combinations of distinct, self contained heuristics can help to improve the execution time and result quality. We expect seeding of improvement heuristics, to result in significantly better performance. That is, we either expect shorter execution times at similar quality, or equal execution times for better quality. The results , however, may vary for the basic heuristic strategies.

Secondly, we introduced a substitution rates based clustering of protein evolution models. The goal is to identify models that are likely to result in similar log likelihood scores, and thereby reducing the number of candidate models. However, our clustering is not able to identify models that yield similar scores. We expect this to be due to the fact that we did not include knowledge of the data in the clustering. We also presented a clustering strategy that seems to be able to overcome this drawback. However, until now we did not exploit this approach.

The approachers of the remaining two sections aim at reducing the effort for likelihood computations. Because, likelihood computations are the dominating task of all PMA heuristics, it would be beneficial to either compute these values faster or safe some computations. We followed two distinct approaches. The one aims at reducing computational demand by exploiting the frequently small changes we apply to model assignments and archiving already evaluated likelihood scores. These strategies are not expected to have any qualitatively influence. The second strategy proposes to predict, the potential of an assignment that is due to be evaluated. If we predict a model assignments of bad quality, we could avoid evaluating it. Thereby, many none promising assignments could be avoided to consume significant computational resources. However, predictions may be wrong. Therefore, actually promising assignments may not be evaluated.

6 Experimental Setup and Results

So far we introduced the PMA problem and developed different strategies to efficiently assign models. Nonetheless, there remain questions that need to be answered. Unfortunately, these cannot be answered at an analytical level only. In this chapter we experimentally evaluate the proposed approaches.

We start by describing the data sets we used, the computer infrastructure, and a method to measure the distance between trees in Section 6.1. Thereafter, the key questions are outlined in Section 6.2. The results of the evaluation are present in Section 6.3.

6.1 Preliminaries

Experimental Data: There does not exist a general benchmark for evaluating phylogenetic approaches. Therefore, in order to provide as general as possible experiments we use a variety of real and synthetic data sets. To reduce the impact of organism-specific properties of real data, we use data from each of the three domains of life. For this purpose we obtained aligned and partitioned data sets from two studies [85, 48]. Table 6.1 summarizes the properties of the data sets.

Domain (abbr.)	Number of species	Number of partitions	Length	Source
Eukaryotes (*euk*)	117	129	37476	Meusemann *et al.* in [48]
Bacteria (*bac*)	992	56	20609	Yutin *et al.* in [85]
Archaea (*arc*)	86	68	17639	Yutin *et al.* in [85]

Table 6.1: Test data sets of each of the three domains of life and their properties.

In addition to real data sets we conducted experiments with synthetic data of varying size. These data sets were simulated for different input trees using the INDELible software [27].

All data sets (as well as the implementations of the heuristics) are available for download at `http://exelixis-lab.org/joerg/pma.tar.gz`.

Computer Infrastructure: Our implementation of the PMA heuristics relies on the POSIX threads based parallel likelihood function of RAxML-Light. We conducted experiments on two different multi core systems with Intel and AMD CPUs. Table 6.2 provides the system specifications.

Because we only use shared-memory parallelization, the majority of experiments (especially the time consuming tasks) were evaluated on the magny system. The nehalem system was mostly used for initial analyses on small data sets.

Codename	CPU	Cores (CPUs)	CPU clock	RAM
Nehalem	Intel Xeon E5530	8 (2)	2.4 GHz	24 GB
Magny	AMD Opteron 6174	48 (4)	2.2 GHz	124 GB

Table 6.2: Computer configurations expiriments were run on.

Tree Distance: To evaluate our approaches we need to compare different tree topologies. For this purpose we apply the commonly used *partition distance* (also called Robinson-Foulds (RF) distance according to the authors of [66]). Note that every branch of a tree separates two sets of taxa (nodes connected to one end of the branch, and those connected to the other end). That is, an edge defines a *bipartition*. The partition distance is defined by the total number of bipartitions that are in only one of the trees. If all bipartitions are identical, the tree topologies are identical as well (RF distance value of 0.0). Usually the number of unshared bipartitions is normalized by the number of possible bipartitioning. Thus, a distance of 1.0 refers to none shared bipartitions (this is the maximum RF distance). The RF distance does not compare branch lengths, that is, as long as the topology is the same the distance is 0.0. For an in-depth explanation of the RF distance see [83].

6.2 Key Questions

In this section we state the objective key questions we want to answer. Firstly, we focus on the importance of the PMA problem, secondly on the performance of the basic heuristics of Chapter 4, and lastly on assessing the performance improvement strategies of Chapter 5.

Does PMA matter? Until now we did not deal with the impact of PMA. In practice it could be the case that random and optimal model assignments yield the same tree. In particular, the naïve heuristics, which can be quickly evaluated, may find model

assignments that generate the same phylogeny as the optimal PMA assignment. In other words, it may not be important to use the optimal PMA. We say that PMA matters, if the naïve model assignment returns suboptimal phylogenies. An additional goal is to assess how frequently the trees differ.

It is obvious that the naïve heuristics return an optimal assignment, if the data only contains one partition. That is, PMA does not matter for such small alignments. If it matters for more than one partition, it would be interesting to know what the impact of PMA is w.r.t. the number of partitions. Not only do we know that the naïve heuristics is optimal for one partition, it is also widely accepted that evolutionary signal increases for more partitions. As the signal increases we suspect the impact of inappropriate model assignments to decrease. We suspect a range of partition numbers for which PMA is of outstanding importance (for which RF distances between ML trees inferred with the true and random assignments is comparatively large).

Which heuristics are suitable? We expect distinct PMA heuristics to perform differently (some may perform poorly). Of course, one wants to use the best search strategy w.r.t. result quality and execution time.

Usually, heuristics are developed to find as good as possible results. Hence, we are interested in the result quality of the heuristic PMA strategies. Heuristic strategies should at least perform better than a random search. We call a heuristic strategy *suitable*, only if it outperforms random search on average (of course the random search must be allowed to consume the same amount of computation resources).

Moreover, we want to know which strategies obtain which quality at which cost. For example, w.r.t. algorithmic complexity, the naïve heuristics clearly outperform the greedy approach (see Sections 4.2.1 and 4.2.2). In other words, it is faster, if the number of included partitions is large. The execution time for improvement heuristics depends on the data. Some algorithms may not even converge in reasonable time for large PMA instances. One would either prefer the heuristic strategy that converges to the best result, the strategy that finds the best current result after reasonable time, or the one that quickly determines a good enough result.

Do optimization approaches accelerate the search? The main goal of the optimization approaches of Chapter 5 is to reduce computational cost. However, because their aim is to quickly guide the search to high-quality areas, they may also improve the quality of results. Some optimizations can also cause "good" assignments—or even larger parts of the search-space—to not be evaluated. Therefore, they may generate

worse results, as well. In order to asses these effects, we implemented the improvements for appropriate heuristic strategies. The results of the "improved heuristics" must be compared to the corresponding results without optimizations. Only if equally good or better results can be obtained an optimization strategy will be useful.

Which approach should be used when? Certainly, for practical usage it is frequently valuable to accept worse solutions in favor of faster execution times. We analyze the trade-off between execution times and result quality. To address this question, it is necessary to use reasonable real data sets. Because, executing the heuristics can be quite time consuming, we assess this question for the most promising approaches only.

6.3 Results

In this section we describe the setup of our experiments and present the results to answer the above questions.

In Section 6.3.1 we start by assessing the importance of PMA. Thereafter, the quality of the basic PMA heuristics is evaluated in Section 6.3.3. Finally, we assess the performance improvement approaches in Section 6.3.4.

6.3.1 Importance of Protein Model Assignment

To show that PMA matters, we show that the naïve heuristics frequently returns suboptimal model assignments. For this purpose we extracted 50 subsamples from each of the data sets in Table 6.1. In order to be able to evaluate the samples exhaustively, the number of partitions was reduced to three. Because, the occurrence of high RF distances depends on the tree size (see [83]), we also subsampled 50 taxa. Thereafter, we computed and compared the optimum PMA and the naïve optimum for each sample. On average, model assignments differed in 57% of all 150 samples. For the archaea data set in 60%, for bacteria, in 58%, and for the eukaryotic samples, assignments differed in 54% of all subsets. Whenever optimal and naïve assignments were different, we conducted a complete ML tree inference (one tree search per assignment). Different model assignments resulted in a identical ML trees in only 14% of the cases. On average, we obtained a RF distance of 9%. Put differently, in 49% of all cases the naïve heuristics produced a different ML tree with an average RF distance of 9%. Moreover, the naïve optimum did never result in a better (i.e., higher likelihood) ML tree than the PMA

optimum.

Furthermore, we also want to provide guidance regarding cases when model assignment is particularly important. However, for this purpose the number of included partitions must be variable. As an exhaustive search is computationally too expensive even for small numbers of partitions, we used simulated data. Therefore, instead of computing to the optimal PMA, we used the "true" model assignment as reference. In particular, we simulated data sets for different tree shapes with 40 taxa and the number of partitions increasing exponentially from 2 to 128. For these data sets, we computed the naïve optimum per data set and conducted a full ML tree search for the—in this case—true model assignment, the naïve assignment, and a random model assignment. Thereafter, we compared the RF distances of these "naïvely" and "randomly" inferred trees to the phylogeny obtained for the true model assignment. Unfortunately, we did not find a dependency between the number of partitions and the accuracy of the naïve heuristics (all RF distances are 0.0). Neither did we resolve a clear relationship between the mismatch (large RF distance in resulting phylogenies) of random and true assignments.

As we already showed that PMA matters, we guess that we could not examine correlations because usually tree inference for simulated data is easier than for real data. This is mainly due to the fact that synthetic data is more regular.

6.3.2 Parameter Setting

In order, to assess the performance of the PMA heuristics, we must first specify their parameters. The amount and type of parameters varies among the heuristics. Table 6.3 summarizes the parameters. For setting the parameters, we differentiate between *on-off*

PMA heuristic	parameters
Greedy	sort/do not sort in advance
Hill-Climbing	steepest/next ascent strategy
SA	Starting temperature $T_0 \in \mathbb{N}$, temperature decrement $\beta \in [0;1]$
GA	Population size $N_{pop} \in \mathbb{N}$, mutation rate $P_m \in [0;1]$

Table 6.3: Parameters and their corresponding domain of the PMA heuristics.

switches and *discrete/continuous parameters*. We start by evaluating the heuristics that only use on-off switches (greedy assignment composition and Hill-Climbing). Thereafter, we assess the parameters for the remaining heuristics (SA and GA)

On-off Switches

Usually, On-off Switches are comparatively easy to specify. Furthermore, greedy assignment composition and Hill-Climbing only have one parameter. For both heuristics these provides a single algorithmic choice. To determine if assignments should be sorted in advance (greedy), or steepest or next ascent should be applied (Hill-Climbing), we evaluate each setting for different data sets (also PMA instances). We can, however, not guarantee that the choices are suitable for any PMA instance.

Data sample	Greedy sorted (1) vs. unsorted (2)		Hill-Climbing next (3) vs. steepest (4)	
$\text{source}_{n\times\#taxa}$	$lnL_{(1)} - lnL_{(2)}$	$t_{(1)}/t_{(2)}$	$lnL_{(3)} - lnL_{(4)}$	$t_{(3)}/t_{(4)}$
$euk_{30\times10}$	-3.65	1.02	0.06	1.43
$arc_{20\times20}$	-3.86	1.10	0.06	1.37
$bac_{20\times20}$	-8.78	1.04	0.18	1.37
$euk_{20\times20}$	32.01	1.10	0.00	1.51
$arc_{30\times20}$	3.03	1.11	-	-
$bac_{30\times20}$	-10.01	1.01	-	-
$euk_{30\times20}$	9.57	1.08	-	-
$arc_{50\times50}$	-20.22	1.05	-	-
$bac_{50\times50}$	-27.34	1.01	-	-
$euk_{50\times50}$	90.25	1.01	-	-

Table 6.4: Results of experimental evaluation of on-off switches for Greedy Assignment Composition and Hill-Climbing. The table summarizes the mean result score and execution time difference.

We extracted multiple data subsamples of 30 partitions and 10 taxa from the eukaryotic data set. Thereafter, we evaluated all algorithmic settings on these samples. Because likelihood scores and execution time for distinct data samples must not be compared to each other, we compare the algorithmic options per data sample. In Table 6.4 we average the per-sample score difference in log likelihood units (note that $a = \frac{b}{c} \Leftrightarrow ln(a) = ln(b) - ln(c)$)) and the execution time ratio per data source and sample size. In this first test, we found that both parameters have comparatively small effect on the result quality. For the greedy heuristic the runtimes were also similar. We obtained comparatively shorter execution times for the steepest ascent strategy, though.

To retrieve more reliable results we evaluated both parameter settings for further data subsample of size 20×20 from data of all three domains. For Hill-climbing we obtained similar results for the additional executions. Therefore, we suspect better performance for the steepest ascent strategy in general. Because for the greedy heuristic the results did not reveal clearly a superior strategy with respect to the result quality, we conducted

two additional evaluations for subsample sizes of 30×20 and 50×50. In summary we can acknowledge the theoretical results of Section 4.2.2 that the greedy heuristic becomes slower for data that is sorted according to partition length. We did, however not identify which heuristic is should be preferred in terms of log likelihood scores.

Nonetheless, we apply the sorting strategy for our remaining evaluations, because it explicitly specifies an order among partitions. For the unsorted greedy heuristic the partition order is just the random order of the data subsamples.

Discrete and Continuous Parameters

Only SA and GA include discrete or continuous parameters. For these heuristics it is difficult to determine good parameter setting because of the infinite parameter domain. Therefore, such parameters cannot be evaluated for all possible settings. To reduce the parameter ranges we apply previous results.

SA and GA include more than one parameter. Because these are not independent many combinations may be applied. Moreover, both heuristics rely on random numbers (e.g., to create candidate assignments). If the random number seed is identical for different executions, for SA, initial assignments are identical, whereas for GA, the individuals of the initial population are. A different seed causes different results, though. Therefore, we repeat every analysis 10 times and average (always removing highest and lowest values) results over the replicates, to reduce random variations.

In [62], Rardin and Uzsoy suggested to retrieve initial parameter configurations by using sufficiently small problem instances. Because these can be evaluated comparatively quick, many parameter configurations may be assessed. Therefore, we extracted test data from the eukaryotes data set, which contains 20 partitions for 10 taxa. For this data set we evaluated SA and GA with a variety of parameter combinations on the nehalem computer configuration (see Table 6.2). Next we describe the experiments for SA, and thereafter the setup for GA.

Simulated Annealing: SA largely depends on the temperature and its decrement. The faster the temperature decreases the faster the acceptance probability decreases. Therefore, the cooling schedule is of major importance for the effectiveness. According to Al-Araidah *et al.* (see [7]) in beginning the temperature should be hight enough to allow almost any neighborhood move. Otherwise SA is likely to end with a solution close to the starting solution. If the temperature is too high, though, SA may degenerates into an almost random search in the initial phase. We approximate the maximum

difference of two neighboring assignments (assignments with a varying model for one partition) on per-partition log likelihood scores. Similar to the clustering corresponding to Figure 5.2, we evaluate the distances between all model per partition, for this purpose. The maximum per-partition distance is then applied as starting temperature.

Typical values for the temperature decrement β lie between 0.8 and 0.99 (see [7]). However, according to Lundy and Mees (see [47]), the temperature should be cooled sufficiently slow, if only one iteration is permitted per temperature. The slower the temperature decreases, the longer the execution time, though. Because the temperature decreases exponentially, we expect the execution time to be exponentially correlated to the β parameter. We evaluated the β cooling parameter for values varying from 0.81 to 0.999 with steps becoming denser from 0.04 to 0.001 between 0.99 and 0.999.

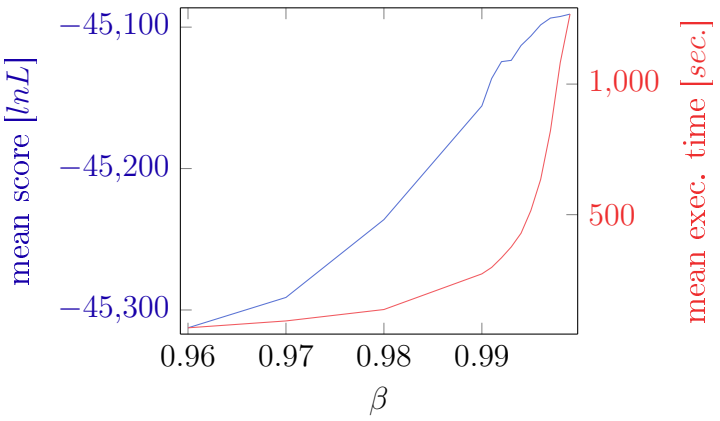


Figure 6.1: SA log likelihood scores and execution time of the β cooling parameter optimization averaged over 18 runs (highest and lowest values were trimmed in advance).

Figure 6.1 summarizes the achieved result qualities and execution time for $\beta > 0.96$. One can observe that, as expected, the execution time increases exponentially with increasing β. We want to use a value, such that, we obtain good results, as quick as possible. $\beta = 0.992$ seems to be a reasonable choice, because we observe a comparatively better likelihood score, whereas the execution time does not yet increase too dramatically.

Genetic Algorithm: We need to provide two parameters as well for the GA—the population size and mutation rate. Obviously, the population size directly influences the execution time (the more individual per generation the higher the chance ti improve upon the currently best score, and consequently less chance to converge). Therefore, we

also expect the scores to become better with larger populations. Common values for the population size lie between 50 and 100.

The influence of the mutation rate is more difficult to predict. Usually quite small values are applied for this purpose. In [33], Haupt *et al.* propose a value between 0.1 and 0.3. However, Haupt *et al.*, expect a binary encoding of solutions, that is, the mutation rate applies to a single bit of the encoding. Because we do not use encoded solutions here, we expect higher mutation rates to be reasonable, as well. Consequently, we evaluate values from 0.1 up to 0.61 (with step size 0.04).

In summary we evaluate $11 \times 15 = 165$ parameter combinations for the GA approach to PMA (11 values for the population size and 15 values for the mutation rate). For this purpose the test data of the parameter optimization of SA was applied.

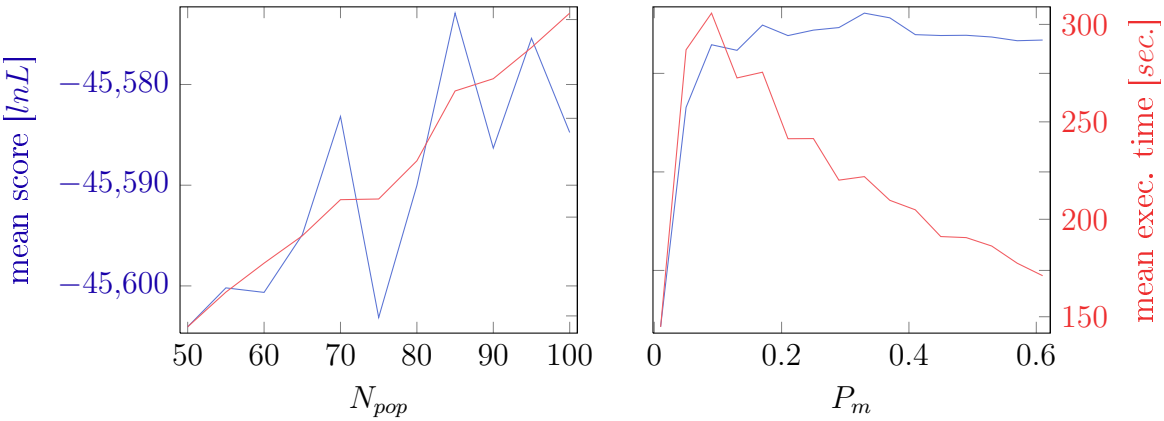


Figure 6.2: GA log likelihood scores and execution time over population size N_{pop} and mutation rate P_m (averaged over 18 runs, excluding the highest and lowest value).

We aggregate the results for the population size and and mutation rate in Figure 6.2. As may be expected, the execution time increases almost linearly with the population size. Moreover, a very low mutation rate (< 0.1) seems to yield bad results. For the remaining rates, likelihood scores vary only slightly. Note however, that result quality is worse for GA if compared to SA (y axes of the corresponding figures are scaled differently). Therefore, we extract the parameters that resolved the best score for GA. That is, we use 0.33 for the mutation rate and a population size of 85 for later experiments.

6.3.3 Suitability of Basic Heuristics

The most simple strategy to find an improved solution is a random search. However, usually more targeted searches perform much better. Therefore, in order to justify the PMA heuristics, we must ensure that the basic heuristic strategies perform better than a random search (given the same computational resources, e.g., execution time or number of assignment evaluations). If a heuristic strategy does not perform better, random search would be equally suitable.

For the purpose of evaluating our strategies, we subsampled 5 alignments (30 partitions and 20 taxa) from each of the data sets from the three domains. Based on these samples, we assessed the suitability of all heuristic strategies. That is, we compared the heuristic result to the random optimum that was found in the same time. GA is the only heuristic that returned worse assignments than the random search in some runs. On average it failed for 9 runs: 2 for eukaryotic samples, 4 for bacteria, and 3 for archaeal samples.

We could have included algorithmic improvements for the suitability assessment, as well. In this case we probably would have found that GA outperforms random search. However, we would expect these different findings to be due to the improvements (i.e., GA remains to be unsuitable for PMA). Therefore, we do not evaluate GA in anymore detail.

6.3.4 Effect of Algorithmic Improvements

In this section we evaluate the improvements introduced in Chapter 5. Firstly, we assess the effect of lazy likelihood computations, secondly combinations of heuristic approaches, and lastly, we evaluate the influence of likelihood pre-scoring.

Lazy Likelihood Computations: We assess lazy likelihood computations before the other techniques, because we expect valuable execution time improvements, with minor influence on the quality of results. Therefore, evaluating the remaining concepts may be accelerated by using lazy likelihood computations.

We evaluate these strategies on the 20 partitions 20 taxa data samples that were already used to assess the suitability of the PMA heuristics. For α and branch length refinement we executed 10 random neighborhood walks of length 100 (evaluation of 100 subsequent neighboring assignments). Each walk was evaluated with and without complete optimization of α and the branch lengths. On average we obtained an acceleration of factor 2.8 by only refining α and the branch lengths instead resetting and

re-optimizing them. The likelihood scores of all evaluated assignments differed by only 0.35 log likelihood units on average.

Assessing the usefulness of the archive is somewhat more difficult, because ideally we would need to know how frequently heuristics re-visit assignments. However, we are mainly interested in an initial assessment of how beneficial an archive can be. Therefore, we simply evaluate a random walk of 1000 assignments for every data sample. To include duplicate assignments, each walk was generated by randomly sampling from a predefined set of 1000 random assignments. Thereafter, we evaluated each walk with and without archiving. We found that for all samples, the time spent to search the archive for already visited assignments was far less than a second on average.

Seeding of Improvement Heuristics: We assessed the effect of seeding improvement heuristics by conducting experiments for various data samples. In particular, we conducted model searches using the Hill-Climbing and SA algorithm, with random initialization, and seeded with the naïve and sorted greedy result (i.e., three different initialization strategies per improvement heuristic and data sample). Table 6.5 summarizes the results of the experiments for Hill-Climbing and SA. We compare the log likelihood values of the seeded heuristic to those of random initialization by determining their difference. Thereafter, we average the differences for per data source and sample size (e.g., $lnL_{(greedy)} - lnL_{(random)}$). In other words, we use the randomly initialized execution as reference. Consequently, if applying a seed improves the likelihood we expect a positive value. Execution time is represented by the ratio of the number of evaluated assignments with seeding to those of random initialization. Therefore, $\#s_{(greedy)}/\#s_{(random)}$ is smaller than one, if less assignments are evaluated by applying a greedy seed.

It can be observed that seeding the Hill-Climbing algorithm only has a minor effect on result quality. The execution time, however, improves. The amount of improvement depends on random variation, (for some data samples random initialization evaluated less assignments). On average seeding reduced the number of evaluated assignments significantly (the naïve seed by ∼15% and the greedy seed by about ∼25%). Therefore, seeding reduces the number of poor assignments in the initial phase. For SA, we obtained better log likelihood scores for all evaluated data samples, by seeding. In average these were better, for the greedy seed. The number of assignment evaluations was not affected. For SA, in contrast, the number of evaluated assignments depends on the starting temperature and its decrement. Therefore, one certainly does not expect major execution time improvements. However, the procedure profits from the seed w.r.t. the result quality.

Data sample	naïve (1) vs. random (3)		greedy (2) vs. random (3)	
$\text{source}_{n\times\#taxa}$	$lnL_{(1)} - lnL_{(3)}$	$\#s_{(1)}/\#s_{(3)}$	$lnL_{(2)} - lnL_{(3)}$	$\#s_{(2)}/\#s_{(3)}$
Seeded Hill-Climbing vs. random initialization				
$arc_{20\times20}$	-0.16	0.72	-0.21	0.59
$bac_{20\times20}$	-0.17	1.05	-0.18	0.93
$euk_{20\times20}$	0.00	0.83	1.35	0.69
Seeded SA vs. random initialization				
$arc_{20\times20}$	12.85	1.0	14.17	1.0
$bac_{20\times20}$	6.83	1.0	9.47	1.0
$euk_{20\times20}$	13.04	1.0	19.67	1.0

Table 6.5: Result quality and "execution time" of seeded Hill-Climbing and SA for different data samples. The execution time is presented by the ratio of the number of assignments of the initialization strategies. All results from either naïve or greedy seed were compared to those with random initialization directly. The rows contain the average per data source and size of the data sample.

Potential Approximation: We already presented an example for which log likelihood scores under joint and per-partition branch length estimates are strongly correlated in Section 5.3.2. This observation forms the basis to approximate the potential of an assignment. Therefore, in order to verify that this property holds for a larger number of PMA instances we evaluated the correlation for 100 random assignments for data subsamples of 20×20 and 50×50 from the data source of Table 6.1. On average we obtained a correlation coefficient of 0.93 (for all data samples the correlation was above 0.82).

From the explanations in Section 5.3.2 we expect that the moving average may be most promising to approximate the potential of an assignment. However, for this measure we need to determine a suitable window size. A larger window size usually causes an increasing amount of assignment evaluations. Furthermore, in an initial evaluation we found that applying the mean score computed with per-partition branch length estimates as a lower bound to estimate the potential of an assignment is critical. This is due to its constant value (it does not depend on progress and performance of the search). If heuristics are initialized with a poor assignment (such that its total per-partition estimated score is comparatively smaller than the average score) it might be the case that all neighboring assignments have poor quality, as well. Therefore, none of these assignments would be evaluated. Consequently, only the accumulated and moving average should be applied to approximate the potential of an assignment.

We evaluate the suitability of the accumulated and moving average for approximating the assignment potential on 10 data subsamples (30 partitions and 20 taxa) from each

real data set of Table 6.1. For this purpose we execute Hill-Climbing and SA for the data samples with varying window sizes of 1, 2, 3, 5, 10, and 15. Thereafter, the results of different window sizes have been compared to those without approximation of the potential (w.r.t. the number of likelihood computations and the log likelihood score of the corresponding best PMA). Table 6.6 summarizes the results by averaging over the source of the data subsamples and the window size. We found that applying the

	$arc_{30\times20}$		$bac_{30\times20}$		$euk_{30\times20}$	
window	ΔlnL	$\frac{\#s_{(1)}}{\#s_{(2)}}$	ΔlnL	$\frac{\#s_{(1)}}{\#s_{(2)}}$	ΔlnL	$\frac{\#s_{(1)}}{\#s_{(2)}}$
SA						
1	10.11	1.97	19.44	2.04	9.59	2.02
2	12.45	1.74	15.98	1.82	13.57	1.78
3	-0.95	1.63	2.10	1.66	-6.57	1.76
5	-6.77	1.55	24.31	1.58	-2.64	1.61
10	3.19	1.48	11.55	1.50	5.94	1.49
15	15.09	1.43	12.10	1.45	8.29	1.44
Hill-Climbing						
1	0.00	2.01	0.23	2.01	0.77	1.80
2	0.01	1.78	0.22	1.82	0.37	1.71
3	0.00	1.80	0.22	1.75	0.00	1.66
5	0.00	1.73	0.22	1.59	0.00	1.55
10	0.00	1.60	0.21	1.67	0.00	1.47
15	0.00	1.55	0.22	1.51	0.00	1.47

Table 6.6: Effects of approximating the potential of assignments on SA and Hill-Climbing. The log likelihood differences of the run without optimization and those of the corresponding heuristic by ΔlnL. Analogously, $\frac{\#s_{(1)}}{\#s_{(2)}}$ represents ratio of the number of likelihood computations without (1) and with (2) the optimization. All values are averaged for 10 different runs.

potential approximation strategy accelerates SA and Hill-Climbing similarly, by a factor between $\sim$1.5 and $\sim$2, mostly depending on the window size. For small window sizes the number of likelihood computations reduces more than for higher window sizes. For Hill-Climbing there is only a very small effect on the obtained result quality, on average. For SA even better assignments were found with a window size of 3. Figure 6.3 depicts an execution of the Hill-Climbing algorithm with and without approximating the potential of assignments. The plot is more smooth if potential approximation is applied, because many the lower values are not evaluated due to their low potential.

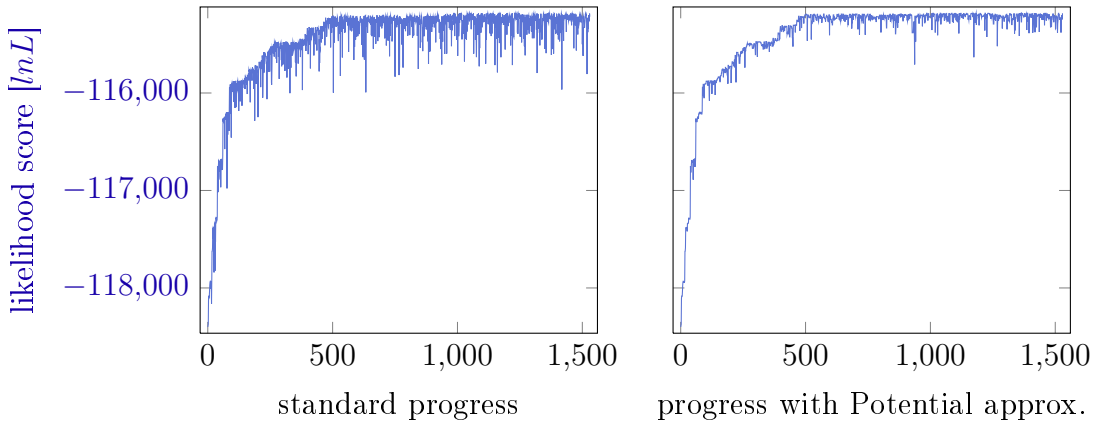


Figure 6.3: Exemplary execution of the Hill-Climbing algorithm without (left) and with (right) approximation of the potential of assignments (window size $w = 5$) on archaea data subsample that are averaged in Table6.6.

6.3.5 Performance Evaluation

Until now we did not comprehensively compare different heuristics to each other. To provide some guidance which heuristic is best under which circumstances, we evaluated the most promising heuristics on data subsamples with 50 partitions and 50 taxa from each data set (5 samples per data set) of Table 6.1 on the magny system. The goal is to compare their performance to each other. During this evaluation we focus on answering which heuristic is either most suitable w.r.t. execution time or the result quality.

For this purpose we executed the naïve heuristics, the greedy assignment composition, as well as SA and Hill-Climbing with all initialization strategies for each sample. Because we know from Section 6.3.4 that usually the randomly initialized versions perform similar or worse, for either result quality or execution time, we first compared the results of the seeded runs of SA and Hill-Climbing to those initialized at random. For Hill-Climbing we found that the naive seed results in shortest execution time on average ($\sim$ 38 minutes). A greedy seed resulted in better likelihood scores, though (232 lnL better than random, and 12 lnL better than with the naïve seed). For the greedy seed the execution time was 45 minutes on average. For SA, greedy and naïve seed executed almost equally long (naïve: 46 minutes, greedy 45 minutes). Also for SA did the greedy seed yield better likelihood scores (on average 22 lnL better). Therefore we we propose to use a greedy seed if improvement heuristics are applied.

In the second step we compare SA and Hill-Climbing with greedy seed to the results obtained by the naïve heuristics and greedy assignment composition. As expected the naïve heuristic is the fastest heuristic (on average it spends only 15% of the mean

execution time of the heuristics). With respect to the likelihood score all heuristics performed quite similarly, whereas on average Hill-Climbing outperformed the other heuristics slightly (with a difference of 6.24 lnL to the averages of the results of all heuristics).

Chapter Summary

Initially, we assessed the importance of the PMA problem in this chapter. For this purpose we compared the naïve model assignment to the exhaustively computed optimal PMA for small data subsamples. On average the model assignments differed in 57%. Out of these cases we obtained and average RF distance of 9% between fully optimized ML trees for the naïve and optimal model assignment. Therefore, we conclude that PMA is important.

To assess the usefulness of our PMA heuristics, we first determined reasonable setting for their parameters. Thereafter, the suitability of the heuristics for the PMA problem has been assessed by comparing each to a random search. The GA is the only only heuristic that returned worse result than the random search in some of the cases. Therefore, we do not expect it to be well suited for PMA.

Furthermore, we evaluated the improvement strategies of Chapter5. We found that refining the values for α and the branches provides significant acceleration. The likelihood scores have been affected only slightly by this optimization. The benefit of the archive depends on how frequently a heuristic may revisits an assignment. Our experiments, however, revealed that searching the unsorted list was faster than a second for distinct random walks of lengths 1000. Therefore, it does not seem important to to use a more comprehensive data structure.

Seeding of the improvement heuristics showed distinct effects. For SA we obtained better likelihood scores comparable time, whereas the execution time could be reduced for the Hill-Climbing algorithm. Further acceleration up to a factor of 2.0 can also be obtained by the approximating the potential of assignments that are to be evaluated.

In summary, we found that the differences of the heuristics w.r.t. the result quality are not very large. Therefore, we suspect the naïve heuristics or greedy assignment composition to be sufficient for most analyses. We did, however, not comprehensively evaluate the impact on the resulting ML trees, yet. If result quality is of major importance Hill-Climbing should be used. However, we also obtained the longest execution times for the Hill-Climbing algorithm.

7 Conclusions and Outlook

This chapter provides the overall conclusions of the thesis. We concentrate on reviewing the results of the thesis. Furthermore, we outline future work.

First we summarize the essential aspects and results of this thesis in Section 7.1. Thereafter, important future directions of future work are described in Section 7.2.

7.1 Summary and Conclusions

In this thesis we introduced and assessed a combinatorial optimization problem called *protein model assignment* (PMA). For this purpose we first outlined some basics of phylogenetics. In particular, we summarized the key concepts of the maximum likelihood method (ML) for phylogenetic inference. Protein-based phylogenetic inference is of outstanding importance, in particular when no DNA data is available or branches are expected to be long (i.e., organisms evolved with many mutations). Furthermore, we explained the usage of explicit stochastic models in ML inferences.

There is no related work that deals with the task of assigning protein models to genetic partitions for the purpose of ML inference with joint branch length estimates. However, we described the task of model selection and its differences from PMA. Thereafter, we presented the formal definition of the PMA problem and discussed its complexity. To efficiently assign protein models we described several heuristics for the PMA problem. In particular, we developed two deterministic constructive algorithms, which construct assignments starting from an initially empty assignment—the greedy assignment composition and the naïve heuristics. Furthermore, we adapted three distinct improvement heuristics—Hill-Climbing, Simulated Annealing (SA), and a Genetic Algorithm (GA). These improve upon a currently best (initially random) assignment by repeatedly applying small changes. Moreover, to improve the performance of these heuristics, we developed several faster approaches by including domain specific knowledge. Finally, we evaluated the PMA heuristics and algorithmic improvements for various data sets.

We found that, phylogenies can significantly differ depending on the applied PMA. Thus, we say that PMA matters. Moreover, the naïve heuristics resolve non-optimal assignments for some data. To provide a guidance on when PMA is particularly important we conducted experiments on synthetic data to assess the impact of the partition number. We could, however, not reveal a relationship for our synthetic data sets. Therefore, it must be assumed, that PMA can be an important issue for all protein-based multi-gene phylogenetic inferences.

By comparing the proposed heuristics to random searches we found that GA does not seem to be appropriate for PMA, because it often returned results of worse quality than those of a random search. These findings may be due to inappropriate parameter settings, though. However, because there exists an infinite amount of values for some GA parameters, it be hard to find a parameter configuration that is suitable.

For the remaining improvement algorithms we showed that omitting full α and branch length re-optimization significantly accelerates likelihood computations. Moreover, avoiding duplicate assignment evaluations by maintaining a simple unsorted list for storing already computed scores, is beneficial. Also seeding the heuristics with good starting assignments either produces comparable result in less time, or results of better quality in comparable time. An approximation of the potential of assignments further accelerates the heuristics.

7.2 Future Work

In this final section we describe directions of future work.

First of all, we want to mention that we assume that intensification should be favored over diversification for PMA heuristics. Therefore, it is promising to further exploit the clustering approach of Section 5.2. By applying this idea it would become possible to identify models for specific partitions that are likely to yield similar scores. Thereby, we could more explicitly differentiate between intensification and diversification steps. Overall we suspect that the more problem specific knowledge we include the better PMA heuristics perform.

From a biological point of view, it seems to be valuable to assess extensions of the PMA problem. As we found that the choice of models is critical w.r.t. resulting phylogenies under joint branch lengths estimates, it is reasonable to believe that this is also the case for proportional branches. The PMA problem does not account for distinct mutation speeds between different genes, though. Proportional branches may be regarded as

a compromise between joint branch length estimation and their estimation on a per-partition basis. Therefore, it would be valuable to assess the impact of assigning mixed models under the proportional branch model.

Lastly, we observed long execution times, especially for large data sets. Until now we exploit only shared memory parallelism for the PMA heuristics. From a technical point of view it is certainly worth to exploit distributed memory approaches for the PMA heuristics, to reduce their execution time for large data sets. Some heuristics, even offer expensive and predictable loops and are therefore predestined for more coarse grained parallelism. For instance, the steepest ascent Hill-Climbing algorithm always evaluates the complete neighborhood before accepting the best neighbor. Therefore, communication is only necessary to map parts of the neighborhood to worker systems and for the workers to return the corresponding best assignment.

Bibliography

[1] E.H.L. Aarts and P.J.M. Laarhoven. Simulated annealing: an introduction. *Statistica Neerlandica*, 43(1):31–52, 1989.

[2] F. Abascal, R. Zardoya, and D. Posada. Prottest: selection of best-fit models of protein evolution. *Bioinformatics*, 21(9):2104, 2005.

[3] F. Abascal, D. Posada, and R. Zardoya. Mtart: a new model of amino acid replacement for arthropoda. *Molecular biology and evolution*, 24(1):1–5, 2007.

[4] J. Adachi and M. Hasegawa. Model of amino acid substitution in proteins encoded by mitochondrial dna. *Journal of Molecular Evolution*, 42(4):459–468, 1996.

[5] J. Adachi, P.J. Waddell, W. Martin, and M. Hasegawa. Plastid genome phylogeny and a model of amino acid substitution for proteins encoded by chloroplast dna. *Journal of Molecular Evolution*, 50(4):348–358, 2000.

[6] H. Akaike. Information theory and an extension of the maximum likelihood principle. In *Second international symposium on information theory*, volume 1, pages 267–281. Springer Verlag, 1973.

[7] O. Al-Araidah, K.A. Shgair, W. Batayneh, and A. Diabat. Efficient approximation of melting temperature in simulated annealing algorithms applied to chebyshev travelling salesman problem. *International Journal of Business Performance and Supply Chain Modelling*, 4(2):145–163, 2012.

[8] D.A. Bader, B.M.E. Moret, and L. Vawter. Industrial applications of high-performance computing for phylogeny reconstruction. In *Proc. of SPIE ITCom*, volume 4528, pages 159–168. Citeseer, 2001.

[9] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR)*, 35(3):268–308, 2003.

[10] M. Boyer, M.A. Madoui, G. Gimenez, B. La Scola, and D. Raoult. Phylogenetic and phyletic studies of informational genes in genomes highlight existence of a 4th domain of life including giant viruses. *PLoS One*, 5(12):e15530, 2010.

[11] Y. Cao, J. Adachi, A. Janke, S. Pääbo, and M. Hasegawa. Phylogenetic relationships among eutherian orders estimated from inferred sequences of mitochondrial proteins: instability of a tree based on a single gene. *Journal of Molecular Evolution*, 39(5):519–527, 1994.

[12] Y. Cao, A. Janke, P.J. Waddell, M. Westerman, O. Takenaka, S. Murata, N. Okada, S. Pääbo, and M. Hasegawa. Conflict among individual mitochondrial proteins in resolving the phylogeny of eutherian orders. *Journal of Molecular Evolution*, 47(3): 307–322, 1998.

[13] L.L. Cavalli-Sforza and A.W.F. Edwards. Phylogenetic analysis. models and estimation procedures. *American journal of human genetics*, 19(3 Pt 1):233, 1967.

[14] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45 (1):41–51, 1985.

[15] A. Cornish-Bowden. Nomenclature for incompletely specified bases in nucleic acid sequences: recommendations 1984. *Nucleic acids research*, 13(9):3021, 1985.

[16] C. Cotta, E.G. Talbi, and E. Alba. Parallel hybrid metaheuristics. *Parallel Metaheuristics*, pages 347–370, 2005.

[17] D. Cuong, G. Olivier, and L. Vinh. Flu, an amino acid substitution model for influenza proteins. *BMC Evolutionary Biology*, 10, 2010.

[18] D. Darriba, G.L. Taboada, R. Doallo, and D. Posada. Prottest 3: fast selection of best-fit models of protein evolution. *Bioinformatics*, 27(8):1164, 2011.

[19] C. Darwin. On the origin of species by means of natural selection london. *J. Murray*, 1859.

[20] M.O. Dayhoff and R.M. Schwartz. A model of evolutionary change in proteins. In *In Atlas of protein sequence and structure*. Citeseer, 1978.

[21] A. de Queiroz and J. Gatesy. The supermatrix approach to systematics. *Trends in Ecology & Evolution*, 22(1):34–41, 2007.

[22] M.W. Dimmic, J.S. Rest, D.P. Mindell, and R.A. Goldstein. rtrev: an amino acid substitution matrix for inference of retrovirus and reverse transcriptase phylogeny. *Journal of Molecular Evolution*, 55(1):65–73, 2002.

[23] A.W.F. Edwards and C.L.L. Sforza. The reconstruction of evolution. *Heredity*, 18, 1963.

[24] B. Everitt, A. Skrondal, and Inc Books24x7. *The Cambridge dictionary of statistics*, volume 4. Cambridge University Press Cambridge, 2002.

[25] J. Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of molecular evolution*, 17(6):368–376, 1981.

[26] J. Felsenstein and G.A. Churchill. A hidden markov model approach to variation among sites in rate of evolution. *Molecular Biology and Evolution*, 13(1):93–104, 1996.

[27] W. Fletcher and Z. Yang. Indelible: a flexible simulator of biological sequence evolution. *Molecular biology and evolution*, 26(8):1879–1888, 2009.

[28] O. Gascuel. Bionj: an improved version of the nj algorithm based on a simple model of sequence data. *Molecular biology and evolution*, 14(7):685–695, 1997.

[29] Duncan Graham-Rowe. Usb stick can sequence dna in seconds. *New Scientist*, 213 (2853):2, 2012.

[30] R.W. Hamming. Error detecting and error correcting codes. *Bell System Technical Journal*, 29(2):147–160, 1950.

[31] M. Hasegawa and S. Horai. Time of the deepest root for polymorphism in human mitochondrial dna. *Journal of molecular evolution*, 32(1):37–42, 1991.

[32] M. Hasegawa, H. Kishino, and T. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *Journal of molecular evolution*, 22(2):160–174, 1985.

[33] R.L. Haupt, S.E. Haupt, and J. Wiley. *Practical genetic algorithms*. Wiley Online Library, 2004.

[34] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89(22):10915, 1992.

[35] J.P. Huelsenbeck and K.A. Crandall. Phylogeny estimation and hypothesis testing using maximum likelihood. *Annual Review of Ecology and Systematics*, pages 437–466, 1997.

[36] J.P. Huelsenbeck, P. Joyce, C. Lakner, and F. Ronquist. Bayesian analysis of amino acid substitution models. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 363(1512):3941–3953, 2008.

[37] D.T. Jones, W.R. Taylor, and J.M. Thornton. The rapid generation of mutation data matrices from protein sequences. *Computer applications in the biosciences: CABIOS*, 8(3):275, 1992.

[38] T. H. Jukes and C. R. Cantor. Evolution of protein molecules. In M. N. Munro, editor, *Mammalian protein metabolism*, volume III, pages 21–132. Academic Press, N. Y., 1969.

[39] T. Keane, C. Creevey, M. Pentony, T. Naughton, and J. Mclnerney. Assessment of methods for amino acid matrix selection and their use on empirical data shows that ad hoc assumptions for choice of matrix are not justified. *BMC evolutionary biology*, 6(1):29, 2006.

[40] S.A. Kelchner and M.A. Thomas. Model use in phylogenetics: nine key questions. *Trends in Ecology & Evolution*, 22(2):87–94, 2007.

[41] B. King. Step-wise clustering procedures. *Journal of the American Statistical Association*, pages 86–101, 1967.

[42] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *science*, 220(4598):671, 1983.

[43] C. Kosiol and N. Goldman. Different versions of the dayhoff rate matrix. *Molecular Biology and Evolution*, 22(2):193–199, 2005.

[44] S.Q. Le and O. Gascuel. An improved general amino acid replacement matrix. *Molecular biology and evolution*, 25(7):1307–1320, 2008.

[45] W.L.S. Li and A.G. Rodrigo. Covariation of branch lengths in phylogenies of functionally related genes. *PloS one*, 4(12):e8487, 2009.

[46] P. Lio and N. Goldman. Models of molecular evolution and phylogeny. *Genome research*, 8(12):1233–1244, 1998.

[47] M. Lundy and A. Mees. Convergence of an annealing algorithm. *Mathematical programming*, 34(1):111–124, 1986.

[48] K. Meusemann, B.M. von Reumont, S. Simon, F. Roeding, S. Strauss, P. Kück, I. Ebersberger, M. Walzl, G. Pass, S. Breuers, *et al.* A phylogenomic approach to resolve the arthropod tree of life. *Molecular biology and evolution*, 27(11):2451–2464, 2010.

[49] T. Müller and M. Vingron. Modeling amino acid replacement. *Journal of Computational Biology*, 7(6):761–776, 2000.

[50] S.B. Needleman, C.D. Wunsch, *et al.* A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[51] D.C. Nickle, L. Heath, M.A. Jensen, P.B. Gilbert, J.I. Mullins, and S.L.K. Pond. Hiv-specific probabilistic models of protein evolution. *PLoS One*, 2(6):e503, 2007.

[52] M. Nirenberg, P. Leder, M. Bernfield, R. Brimacombe, J. Trupin, F. Rottman, and C. O'Neal. Rna codewords and protein synthesis, vii. on the general nature of the rna code. *Proceedings of the National Academy of Sciences of the United States of America*, 53(5):1161, 1965.

[53] S. Pääbo *et al.* The mosaic that is our genome. *Nature*, 421(6921):409–412, 2003.

[54] H. Pearson. Genetics: what is a gene? *Nature*, 441(7092):398–401, 2006.

[55] Jonathan Pevsner. *Bioinformatics and functional genomics*. Wiley-Blackwell, Hoboken, NJ, 2. ed. edition, 2009. ISBN 978-0-470-08585-1. Includes bibliographical references and index.

[56] D. Posada and T.R. Buckley. Model selection and model averaging in phylogenetics: advantages of akaike information criterion and bayesian approaches over likelihood ratio tests. *Systematic Biology*, 53(5):793, 2004.

[57] D. Posada and K.A. Crandall. Modeltest: testing the model of dna substitution. *Bioinformatics*, 14(9):817–818, 1998.

[58] D. Posada and K.A. Crandall. Selecting the best-fit model of nucleotide substitution. *Systematic Biology*, 50(4):580, 2001.

[59] T. Pupko, D. Huchon, Y. Cao, N. Okada, and M. Hasegawa. Combining multiple data sets in a likelihood analysis: which models are the best? *Molecular biology and evolution*, 19(12):2294, 2002.

[60] A. Purvis. A composite estimate of primate phylogeny. *Philosophical Transactions of the Royal Society of London. B: Biological Sciences*, 348(1326):405, 1995.

[61] S. Rana, A.E. Howe, L.D. Whitley, and K. Mathias. Comparing heuristic, evolutionary and local search approaches to scheduling. In *Third Artificial Intelligence Plannings Systems Conference (AIPS-96)*. Citeseer, 1996.

[62] R.L. Rardin and R. Uzsoy. Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, 7(3):261–304, 2001.

[63] J.H. Reeves. Heterogeneity in the substitution process of amino acid sites of proteins coded for by mitochondrial dna. *Journal of molecular evolution*, 35(1):17–31, 1992.

[64] S. Renaud, P. Chevret, and J. Michaux. Morphological vs. molecular evolution: ecology and phylogeny both shape the mandible of rodents. *Zoologica Scripta*, 36 (5):525–535, 2007.

[65] J. Ripplinger and J. Sullivan. Does choice in model selection affect maximum likelihood analysis? *Systematic Biology*, 57(1):76–85, 2008.

[66] DF Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53(1-2):131–147, 1981.

[67] O. Rota-Stabelli, Z. Yang, and M.J. Telford. Mtzoa: A general mitochondrial amino acid substitutions model for animal evolutionary studies. *Molecular phylogenetics and evolution*, 52(1):268, 2009.

[68] P.J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[69] G. Schwarz. Estimating the dimension of a model. *The annals of statistics*, 6(2): 461–464, 1978.

[70] J. Shallit. What this country needs is an 18¢ piece. *Mathematical Intelligencer*, 25 (2):20–23, 2003.

[71] A. Stamatakis. *Distributed and parallel algorithms and systems for inference of huge phylogenetic trees based on the maximum likelihood method*. PhD thesis, Technische Universität München, Universitätsbibliothek, 2004.

[72] A. Stamatakis, T. Ludwig, and H. Meier. Raxml-iii: a fast program for maximum likelihood-based inference of large phylogenetic trees. *Bioinformatics*, 21(4):456, 2005.

[73] A. Stamatakis, P. Hoover, and J. Rougemont. A rapid bootstrap algorithm for the raxml web servers. *Systematic biology*, 57(5):758–771, 2008.

[74] A. Stamatakis, A.J. Aberer, C. Goll, S.A. Smith, S.A. Berger, and F. Izquierdo-Carrasco. Raxml-light: A tool for computing terabyte phylogenies. *Exelixis-RRDR*, 3, 2012.

[75] A.S. Tanabe. Kakusan: a computer program to automate the selection of a nucleotide substitution model and the configuration of a mixed model on multilocus data. *Molecular Ecology Notes*, 7(6):962–964, 2007.

[76] A.S. Tanabe. Kakusan4 and aminosan: two programs for comparing nonpartitioned, proportional and separate models for combined molecular phylogenetic analyses of multilocus sequence data. *Molecular Ecology Resources*, 2011.

[77] S. Veerassamy, A. Smith, and E.R.M. Tillier. A transition probability model for amino acid substitutions from blocks. *Journal of Computational Biology*, 10(6): 997–1010, 2003.

[78] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.

[79] S. Whelan and N. Goldman. A general empirical model of protein evolution derived from multiple protein families using a maximum-likelihood approach. *Molecular Biology and Evolution*, 18(5):691–699, 2001.

[80] C.R. Woese and G.E. Fox. Phylogenetic structure of the prokaryotic domain: the primary kingdoms. *Proceedings of the National Academy of Sciences*, 74(11):5088, 1977.

[81] D.H. Wolpert and W.G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.

[82] Z. Yang. Maximum-likelihood estimation of phylogeny from dna sequences when substitution rates differ over sites. *Molecular Biology and Evolution*, 10(6):1396–1401, 1993.

[83] Z. Yang. *Computational molecular evolution*. Oxford University Press, USA, 2006.

[84] Z. Yang and B. Rannala. Bayesian phylogenetic inference using dna sequences: a markov chain monte carlo method. *Molecular Biology and Evolution*, 14(7):717–724, 1997.

[85] N. Yutin, P. Puigbò, E.V. Koonin, and Y.I. Wolf. Phylogenomics of prokaryotic ribosomal proteins. *PloS one*, 7(5):e36972, 2012.

[86] G. Zapfel, G. Zäpfel, R. Braune, and M. Bögl. *Metaheuristic Search Concepts: A Tutorial with Applications to Production and Logistics*. Springer Verlag, 2010.