Evolutionary Tree Reconstruction and its Applications in Protein Classification

Róbert Busa-Fekete

Research Group on Artificial Intelligence

June 2008

A thesis submitted for the degree of doctor of philosophy
of the University of Szeged

University of Szeged
Doctoral School in Mathematics and Computer Science
Ph.D. Programme in Informatics

Preface

In this thesis we are mainly concerned with the study of phylogenetic tree reconstruction algorithms and their use in protein classification. In the first part of the thesis we present two tree reconstruction methodologies. Then, we investigate how we can utilize of tree reconstruction methods in automatic sequence classification.

The highly accurate distance-based tree reconstruction methods are very important in sequence analysis. For example, a multiple-sequence alignment can be aided by an accurate phylogenetic tree. Here we introduce a distance-based method which is based on the least-squares criteria. In many test scenarios it can prove superior compared to the traditional ones. After we will introduce a consensus tree method, which in biological evolutionary studies is a commonly used technique for representing a collection of phylogenetic trees by a single tree. The algorithms used for this are known as the consensus tree methods. In this thesis the main method we will apply is called the *Maximum Clique Consensus* approach. In many tasks this approach is found to be highly efficient in tree reconstruction compared to other widely-used consensus tree methods.

In the second part of the thesis we will investigate the application of the phylogenetic tree reconstruction methods in protein classification. The problem of protein sequence classification is one of the crucial tasks in the interpretation of genomic data. Many high-throughput systems were developed which seek to categorize the proteins based just on their sequences. However, modelling how the proteins have evolved can also be useful in the task of classifying sequenced data. Hence phylogenetic analysis has grown in importance in the field of protein classification. This approach not only relies on similarities in sequences, but it also takes into account phylogenetic information stored in a tree (e.g. in a phylogenetic tree). Eisen first used phylogenetic trees in protein classification, and his work has revived the discipline of phylogenomics. Here we shall focus on the application of distance-based phylogenetic tree reconstruction methods in automatic protein classification. We will introduce these kinds of algorithms to tackle a wide range of biological classification problems. We then justify the practical usefulness of our techniques in experiments involving large, biological protein classification benchmark datasets.

Róbert Busa-Fekete, May 2008.

Acknowledgements

Acknowledgements First of all, I would like to thank my supervisors, Prof. János Csirik and Dr. Adrás Kocsor for supporting my work with useful comments and letting me work at an inspiring department, the Research Group on Artificial Intelligence. I would also thank all my colleagues/friends in alphabetical order: András Bánhalmi, Richárd Farkas, Attila Kertész-Farkas, Róbert Ormándi and Győrgy Szarvas.

Thank you to the collaborators for giving me new ideas and listening my talks with great interest: Sándor Pongor, (International Centre for Genetic Engineering and Biotechnology), Csaba Bagyinka, (Biological Research Center, Hungarian Academy of Sciences), Tivadar M. Tóth, (University of Szeged).

I would also like to thank David Curley for scrutinizing and correcting this thesis from a linguistic point of view.

I would like also to thank my Böbe for the love and happiness she has brought to my life. Last, but not least I wish to thank my parents, my grandparents and my brother for giving me their unlimited love and support. I would like to dedicate this thesis to them.

Notation used

$\mathbb{N}$	natural numbers
$\mathbb{R}$	real numbers
$\mathbb{R}_+$	positive reals
T	phylogenetic tree
$\mathcal{T}_T$	taxon set of T
L_i	leaf of a phylogenetic tree which represents the ith taxa
$T^{\mathcal{C}}$	the cluster set of the phylogenetic tree T
e_T	distance estimation error of a phylogenetic tree T
D^T	patristic distance or leaf distance of T
$d(x, y)$	evolutionary distance of x and y
Q	generator matrix
$P(t)$	probability transition matrix
$IC(.)$	insertion cost
$s(x_i, y_j)$	similarity value of x_i and x_j
S	similarity matrix

List of Figures

List of Tables

Contents

Chapter 1

Introduction

For over a hundred years the theory of evolution has been the most widely-accepted model of how species evolve and have evolved. The discipline which deals with the modelling of evolution is called phylogenetics (the word originates from the conjunction of the Greek words phyle meaning tribe or race and genesis meaning birth or beginnings). The methods which are most widespread in phylogenetics represent the process of species evolution with a so-called phylogenetic tree, which corresponds to a weighted tree-graph where the leaves represent the biological objects of interest. The reconstruction of these kinds of trees give rise to several problems which are interesting from both a computer scientific and biological point of view.

Earlier phylogenetics just focused on the evolution of species based on morphological characteristics, but nowadays the explosive advancement in molecular biology now requires the investigation of proteins as well. The wealth of sequenced protein data allows us to perform novel examinations on them. The possibility of comparing protein sequences has pushed research towards the systematization of proteins isolated from distinct species. Proteins that share a high sequence identity or similarity foster the presumption that they share a common ancestor, and therefore we call them evolutionary related or homologous proteins. The analysis of evolutionary related proteins has also become a key issue in phylogenetics. After our brief introduction we can state the basic goal of phylogenetics: it is to reconstruct an appropriate tree topology based on protein sequences that have high sequence similarity. We should mention here that the high sequence similarity of proteins usually implies that they share a common functionality too, but this is not always the case.

In a broader sense this thesis concentrates on two key fields, namely artificial intelligence and bioinformatics. Within these fields we focus on evolutionary tree reconstruction and machine learning. As the thesis consists of two parts, the author's results will also be divided into two parts. The *first part* of the dissertation includes an introduction to evolutionary tree reconstruction methods.

Several tree building method have been worked out and some of them are now widely used, like Neighbor Joining (NJ) [9] and the Unweighted Pair Group Method with Arithmetic mean (UPGMA) [10]. These methods form part of the so-called distance-based or distant matrix methods, because they reconstruct the evolutionary history

of biological objects based only on a pre-determined value or observed distance value among them. Methodologically speaking, our Multi-Stack (MS) [11] algorithm also belongs to this field. Expressed in simple terms, the MS method finds a weighted tree topology that predicts the observed set of distances as closely as possible. To be more precise, a weighted tree defines a distance value for all pairs of leaves – i.e. the sum of the weights of edges containing the path between them. Hence we expect from its output tree that the distance values defined by itself differ from the observed distances as little as possible. To find this tree is an NP-complete problem when we have an arbitrary distance measure [12], thus it can be applied only to heuristical solutions. The idea behind the MS method is to build an optimal tree for the subsets of the proteins of interest, and then it will join these subtrees in an iterative manner. We can efficiently apply this bottom-up approach in many test scenarios, and the MS method often appears to outperform many traditional tree building methods.

Since there are many tree building methods available which produce more than one possible evolutionary history, or the distinct tree building methods reconstruct different trees, in many cases it is necessary to use those methodologies which are capable of reconstructing one "representative" tree based on many different phylogenetic trees. These kinds of methods are the consensus tree methods [13], and they are usually applied in the last step of phylogenetic analysis.

In general, all the interior points in a rooted phylogenetic tree determine a subset of the biological object of interest (i.e. the objects which are represented by those leaves in the tree which can be found below the inner point). Noting this fact, we can say that the concept of a phylogenetic tree and the concept of a hierarchical set system are essentially equivalent. The hierarchical set systems consist of those subsets or, in other words, clusters which are pairwise compatible. Hence each phylogenetic tree corresponds to a pairwise compatible cluster set. Most of the consensus methods determine a compatible subset of the cluster sets of the input trees in different ways, based on the cardinality of cluster occurrences in the input trees. The computation required can be done in polynomial time. Our goal here is to find the subset of the input clusters where the total number of cluster occurrences is maximal. Moreover, we can also define an arbitrary (not necessarily occurrence-based) weighting function on the clusters of the input trees. We have solved this consensus tree building approach efficiently [14], and we showed that it can perform a more precise phylogenetic analysis than the traditional consensus methods like Majority-Rule, Strict and Greedy consensus [15].

The topic of the *second part* of the thesis is the application of tree building methods in protein classification. The automated protein classification is a crucial task in the today's fields of biology research. The unknown genes/proteins of distinct organisms can be retrieved and stored in the form of character sequences that are several hundred characters in length. Nowadays, it has become routine to compare this data with the sequences of known proteins/genes using a method based on a *approximate string matching* technique. Then applying a machine learning method the unknown protein can be placed into some known category (e.g. a structural or functional one)[1]. The

automated data annotation system of the often mentioned genome research project is based on this methodology.

In our studies we strive to develop novel, efficient protein classification algorithms. Our basic assumption is that the structure of the datasets can be represented by a phylogenetic tree, and that by using this representation protein classification can be made significantly more efficient [16; 17]. The protein classification methods, which also make use of phylogenetic information, belong to the domain of phylogenomics [18]. Hence our methods may be regarded as tools for this field as well.

1.1 Summary by Chapters

This thesis is comprised of two main parts. The first part (chapters 2-5) discusses two phylogenetic reconstruction algorithms and evolutionary models, while the second (chapters 6,7 and 8) deals with the application of the phylogenetic tree reconstruction method and the application of ROC analysis in protein classification (Chapter 9).

There are two chapters in the first part which do not contain scientific contributions from the author, but have the goal of introducing the basic notations and models we used. In Chapter 2 we introduce the mathematical tools applied in current models. Then we give a brief overview of sequence evolution models which are necessary for understanding the notation employed for the evolutionary distances. After, in Chapter 3 we also review the tree evaluation criteria which are widely used in phylogenetic analysis.

In Chapter 4 we introduce our Multi-Stack based phylogenetic tree reconstruction method. We then provide a comprehensive comparative test. In the next section, we give a short overview of the consensus tree methods which are most commonly used in the bioinformatics community, and we also introduce a novel consensus tree technique that is based on a combinatorial optimization problem.

The second part of this thesis focuses on an investigation of tree-based protein classification methods. In Chapter 6 we describe the classification problem from a machine learning point of view. We also provide a brief description of the datasets we used in our tests and the application of model evaluation techniques which are commonly used in protein classification such as Receiver Operator Characteristic (ROC) analysis.

Next in Chapter 7 we present two simple algorithms where we made use of a distance-based phylogenetic tree building method in protein classification. The analysis of these methods leads us to suggest an alternative propagational approach. These kinds of methods are described in Chapter 8.

1.2 Summary by Results

In the following a thesis-like listing of the most important results of the dissertation is given. Table 1.1 shows which thesis by the author is described in which publication.

	[11]	[14]	[17]	[16]	[19]
I. (a)	•				
I. (b)		•			
I. (c)	•	•			
II.(a)			•		
II.(b)				•	
II.(c)					•

Table 1.1: The relation between the theses and the corresponding publications

I. (a) *The author developed a Multi-Stack based phylogenetic tree building method which makes use of least-squares criteria. In this way he produced a novel algorithm which is competitive with the widely used distance-based tree building methods, and it can reconstruct the evolutionary history of those datasets in a better way where the biological objects (sequences of interest) have lower similarity [11]. This improvement can be shown using evolutionary distances as well as using alignment-free sequence distances. In addition, the MS method achieve a better results in many test scenario than the Fitch-Margoliash algorithm which is also based on the least-squares criteria. (Chapter 4)*

(b) *The author solved the* Max Clique Consensus *problem via a binary integer programming task. With this approach an arbitrary weighting of subsets one can find the compatible subsets that have maximal weights. In addition, the author introduced a novel Maximum Likelihood weighting scheme, which leads to an efficient phylogenetic reconstruction technique. He tested this method with different evolutionary models and found that this approach in many case outperforms the widely used consensus tree building methods [14]. The trees in the tests were generated by the widely-used PAUP program package[20], and the consensus methods were compared to each other on these trees. Moreover, the author also compared the consensus methods on a real-life database. (Chapter 5)*

(c) *The author provided a testing framework where the different phylogenetic reconstruction techniques could be compared using different evolutionary models over a wide range [11; 14]. In this testing methodology the biological sequences (DNA or protein) are generated based on a predetermined model evolutionary tree. Next, on this set of sequences the tree-building method of interest are applied, and it produces an output tree, which will be compared to the predetermined model tree. Based on the similarity of these trees we can estimate the accuracy of the particular tree reconstruction method. This testing framework provides a more comprehensive testing environment than the bootstrap method [21] because in this framework we can investigate the efficiency of the tree-building method using different evolutionary models. (Chapters 4 and 5)*

II. (a) *The author introduced the TreeInsert and TreeNN methods, which are novel tree-based protein classification algorithms. In contrast to the earlier methods, the algorithms he introduced here make use of just the sequence similarities. Thus they are readily applicable in a wide range on protein classification tasks. The author compared the tree-based methods on many protein classification tasks using ROC analysis, and they were often significant better. The experiments showed that it is worth applying phylogenetic information in protein classification. [17]. (Chapter 7)*

(b) *The author devised two tree-based propagational methods, namely TreeProp-N and TreeProp-E. These methods may be regarded as extensions of TreeNN, because all of these methods update the sequence similarities using the topology of a phylogenetic tree. In experiments these propagational algorithms usually gave a better performance in protein classification comparing to the former systems [16]. (Chapter 8)*

(c) *The author created a ROC analysis-based evaluation method which is a more reliable model evaluation technique than the original ROC analysis when the distribution of the classes is imbalanced. Applying it, a model selection could be carried out more reliably than with the other approaches[19]. He tested this approach on several large-scale datasets. (Chapter 9)*

Part I

Phylogenetic tree reconstruction

Chapter 2

Background and Notation

2.1 Graphs, Trees, Phylogenetic trees

Before we introduce the concept of a phylogenetic tree, we first need to introduce some basic graph theoretical notations, because graphs play an important role in phylogenetics. The notation we used is borrowed from [22].

An undirected graph G is an ordered pair (V, E) consisting of a non-empty set V of vertices and a multiset E of edges, each of which is an element of $\{\{x, y\} : x, y \in V\}$. Here x and y are the endpoints of an edge. Two vertices in G are said to be adjacent if there is an edge between them. An edge is a loop if its endpoints are the same. Edges that join the same distinct pair of vertices are called parallel edges. A simple graph is a graph without loops and parallel edges. Two edges are adjacent if they share a common endpoint. We shall denote the set of vertices and edges of G by $V(G)$ and $E(G)$, respectively. A subgraph of a graph G is a graph whose vertex and edge sets are subsets of those of G.The degree of a $v \in V(G)$, denoted by $deg(v)$, is the number of edges whose endpoint is v.

A path in graph G is a sequence of distinct vertices $v_1, v_2, \ldots, v_k$ such that, for all $i \in \{1, 2, \ldots, k-1\}$, v_i and v_{i+1} are adjacent. In addition, if $v_1 = v_k$, then the path is said to be a cycle. If a graph does not contain cycle, then we say it is acyclic. A graph is connected if each pair of vertices in G can be joined by a path. The length of the path is the number of edges it contains. The length of a path will be denoted by $p(v_1, v_k)$ between the vertices v_1 and v_k.

Two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are isomorphic if there is a bijection between their vertex set such that it preserves their adjacency.

A graph which has an associated weight function with its edges is said to be a weighted graph. This weight function w usually assigns real numbers to the edges.

With these definitions above we can now define the concept of a tree.

Definition 2.1 *A tree t is a connected acyclic simple graph.*

The vertex set of a tree t of degree one is said to comprise the leaves of t. The vertex of t that is not a leaf is called an interior vertex. A tree is binary if every interior

vertex is of degree three (except the root, if the tree is rooted). Moreover, a connected subgraph of a tree t is called a subtree of t.

In this thesis we shall consider only those weighted trees where the weight function takes only positive real values, that is, the weight function $w : E(t) \to \mathbb{R}^+$. A weighted tree assigns a distance for each pair of leaves (– calculated by summing the weights of the edges on the unique path between them) that is called the *leaf* distance of t, and will be denoted by D^t.

After the introduction of the concept of a tree, we can simply characterize the tree by an equality based on the number of vertices and the number of edges. This theorem and its proof can be found in [22].

Theorem 2.1 *Let $G = (V, E)$ be a graph. Then the following are equivalent:*

i.) G is a tree;

ii.) for any two vertices v_1 and v_2 in V there exists a unique path in G from v_1 to v_2;

iii.) G is connected and $|V| = |E| + 1$.

Definition 2.2 *Let X be a set of n taxa. An* X-tree *T is an ordered pair $T = (t, \phi)$, where t is a tree and $\phi : X \to V(t)$ is a bijection. If we restrict ϕ to label only the leaves of t then we obtain the definition of the* phylogenetic X-tree *or simply* phylogenetic tree.

A phylogenetic tree is a rooted phylogenetic tree if it has a vertex $r \in V$ which is of degree two. The vertex r is called as the root of the phylogenetic tree. In addition, if every interior vertex of t is of degree three, then T is a *binary phylogenetic tree*. If we regard t as a rooted tree, then there is only one internal node of degree 2; otherwise the degrees of the internal node are always 3. Here we will deal only with rooted phylogenetic trees. The subset of the descendants of an internal node is called as a *monophyletic group* or *cluster*, the internal nodes being the *most recent common ancestor* of the *monophyletic group* or *cluster*. Thus the internal nodes of a phylogenetic tree and the clusters are equivalent concepts. This way each phylogenetic tree corresponds to a set of compatible subsets $\mathcal{C}$ (i.e. for all $A, B \in \mathcal{C}$ either $A \subseteq B$, or $B \subseteq A$, or $A \cap B = \emptyset$). This construction is also known as a *Linnean Hierarchy*. We will denote the clusters of T by $T^{\mathcal{C}}$, and the taxon set of a T phylogenetic tree by $\mathcal{T}_T$, which corresponds to X. The leaf which represents the $\mathcal{X}_i \in \mathcal{T}_T$ taxon will be denoted by L_i.

As regards the cluster sets of the phylogenetic trees, it is possible to define a partial ordering $\preceq$ on the phylogenetic trees of n leaves in a natural way.

If $T_1^{\mathcal{C}} \subseteq T_2^{\mathcal{C}}$ holds for two cluster sets then the T_1 tree is a refinement of the T_2 tree. This relation will be a partial ordering in the tree space and the minimal elements of $\preceq$ will be exactly binary phylogenetic trees because they have no refinement.

The Robinson-Foulds (RF) distance or *symmetric difference* for rooted trees [23] is based on this approach as well. Because the RF distance of two rooted phylogenetic

trees T_1 and T_2 represents the cardinality of the symmetric difference of their cluster set, we have

$$RF(T_1, T_2) = \left|\left(T_1^{\mathcal{C}} \triangle T_2^{\mathcal{C}}\right)\right| = \left|\left(\left(T_1^{\mathcal{C}} \backslash T_2^{\mathcal{C}}\right) \cup \left(T_2^{\mathcal{C}} \backslash T_1^{\mathcal{C}}\right)\right)\right|. \tag{2.1}$$

If we regard T_1 as a *reference tree* or *origin tree* and T_2 as an estimated tree of T_1, the first term in this equation can also be considered as the false negative cluster set and the second term as the false positive cluster set. In accordance with this, the *false negative rate* is the percentage rate of the false negative clusters in $T_1^{\mathcal{C}}$, and the *false positive rate* is defined in an analogous way. Thus

$$FNR = \frac{\left|T_1^{\mathcal{C}} \backslash T_2^{\mathcal{C}}\right|}{\left|T_1^{\mathcal{C}}\right|}, \qquad FPR = \frac{\left|T_2^{\mathcal{C}} \backslash T_1^{\mathcal{C}}\right|}{\left|T_2^{\mathcal{C}}\right|}. \tag{2.2}$$

There is also an extension of the RF distance known as the Branch Score Distance (BSD given in [24]). BSD not only takes into account the cardinality of different clusters, but it also takes into account the edge lengths of the phylogenetic trees. So this will provide more information about the differences between tree topologies.

2.2 Interpretation of a phylogenetic tree

The construction of rooted phylogenetic trees provides a convenient representation of evolutionary relationship in biology. For example, the hierarchical classification that was firstly mentioned by Darwin can be easily represented as a tree structure. Moreover, we can express sequential evolutionary events as well, because we can assume that time flows along the tree away from the root, that the root corresponds to the common hypothetical ancestor, and that the inner points correspond to other hypothetical ancestors. These are called Hypothetical Taxonomic Units (HTUs). Here the leaves of the tree represent the extant biological object of interest. They can be species, proteins, genes or genomes. The objects which are represented by the leaves of a phylogenetic tree are called Operational Taxonomic Units (OTUs).

In the case when the leaves represent genes, the phylogenetic tree is called a gene tree. The interior vertices can be mapped to particular biological events such as gene speciation and gene duplication [25]. But if the leaves represent species then the interior vertices correspond to clusters of species.

2.3 Finding the best tree

Before we attempt to discover which tree is better than another one, it is worth counting up the trees that have n leaves. With this calculation we can get an insight into

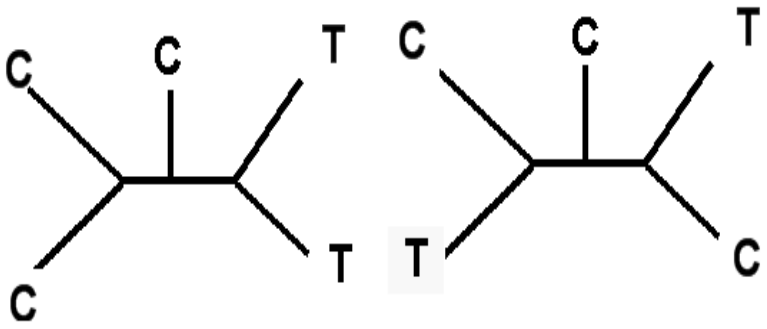


Figure 2.1: A simple example of tree evaluation using the parsimony criterion.

the difficulties of phylogenetic tree reconstruction, and we can exclude some basic approaches, like exhaustive search methods.

Theorem 2.2 *(Felsenstein, [26]) Let us denote the set of rooted binary phylogenetic trees having n leaves by $RB(n)$, and the cardinality of this set by $|RB(n)|$. Then following formula holds:*

$$|RB(n)| = (2n-3)!! \tag{2.3}$$

This tells us that the number of rooted binary phylogenetic trees grows super-exponentially with the number of leaves, so there is no hope of enumerating all of the trees and evaluating them. This simple fact suggests, that we should try an alternative search technique and heuristic.

The next question which arises is how we should evaluate a tree. In this thesis we shall just deal with those trees whose leaves represent proteins. So we can rephrase this question and ask how a phylogenetic tree fits into the phylogeny of a set of sequences. There are many tree criteria in use which could help us answer this question. Using these criteria, we can evaluate different trees if we have a given set of sequences. We should mention here the pioneering work of Edwards and Cavalli-Sforza [27] who first formulated the notion of the parsimonious approach. This means that a tree is to be preferred if the biological changes/events (for example, mutation) along its edges are as few as possible. Illustrating this approach, let us consider the trees depicted in Figure 2.1. In this example the leaves of the trees represent proteins whose sequences are at length one. Because we can assign sequences to the interior nodes of the left tree so that there is a biological change along only on one single edge, we intuitively prefer the left one. While on the right tree there are at least two biological changes along its edges. In the next section we shall give an overview of the tree evaluation criteria, and their efficient mathematical handling.

Chapter 3

Evolutionary models, evolutionary distances and tree criteria

Modelling the evolution of sequences is a fundamental task of bioinformatics. But instead of carrying out a comprehensive overview of this field we will just introduce those models and approaches that we actually used in our studies.

3.1 Sequence alignment

In a biological context the changes which occur in a sequence are called mutations, and they are usually classified in the following categories: insertion, deletion, substitution. So when we consider two sequences which share common ancestor sequences or, in other words, evolved from a common ancestor, then an alignment is basically the search for the possible locations of the different mutations.

We can distinguish two different alignment types which depend on the number of sequences in the alignment: pairwise alignment and multiple alignment.

If we want to formally describe the alignment methods, we first need to state some standard definitions. A DNA sequence is a string over the alphabet Σ which contains four letters A, C, G and T representing four distinct nucleotides. Protein sequences are over an alphabet of 20 letters, each representing a unique amino acid. A multiple alignment of k sequences is obtained by inserting spaces in the sequences such that the sequences have the same length l and they can be arrayed in k rows of l columns each. In the case of $k = 2$ it is said to be a pairwise alignment. A space (or gap) will be denoted by Δ and will be regarded as a new letter over the alphabet $\Gamma = \Sigma \cup \Delta$. Given two sequences s_1 and s_2 in the alignment, then each letter σ of s_1 is in the same column of a letter of s_2; we then say that σ is opposite to a unique letter of s_2. A match occurs where two identical letters are opposite each other in the two sequences s_1 and s_2; otherwise two non-identical opposing letters give a mismatch which is viewed as a replacement. The insertion of a space in a sequence opposite a letter σ of a second sequence is viewed as the deletion in the first sequence of the letter σ or an insertion of σ into the second one. A score d is assigned to each pair of letters and it is generally

described by means of a $|\Gamma| \times |\Gamma|$ symmetric matrix. These scoring schemes can be represented in a so-called scoring matrix. The most commonly used matrices are the BLOSUM matrix family [28] for amino acids and the PAM matrix family for nucleic acids.

After, we can define the value of a multiple sequence alignment in a simple way by summing the scores of all the columns in a multiple alignment, where the score of each column is the sum of the scores of all distinct unordered pairs of letters in the column. An optimal multiple alignment of a set of sequences is the one that minimizes the value over all possible alignments. Unfortunately, it has been proved that to find the optimal multiple alignment for k sequences is NP-hard when $k \geq 3$ [29]. The scores of the multiple alignment of $s_1, s_2, \ldots, s_n$ shall be denoted by

$$d_a(s_1, s_2, \ldots, s_n) = \sum_{i=1}^{k} \sum_{j=1}^{l-1} \sum_{t=j+1}^{l} d\left(s_j^i, s_t^i\right), \tag{3.1}$$

where s_i^k is the kth letter of the ith sequence, and d is the scoring function over $\Gamma \times \Gamma$.

The problem of multiple sequence alignment can be aided by a phylogenetic tree which has been reconstructed using pairwise distances of sequences of interest [30]. Having an accurate phylogenetic tree for the sequences of interest can speed up a multiple sequence alignment and make it more accurate. This area is a hot topic nowadays in bioinformatics because, based on an accurate sequence alignment, we can carry out a domain search or a highly accurate protein classification.

Pairwise alignment methods can be also classified into two main groups, namely the global alignment methods and the local alignment methods. If we attempt to align whole sequences then we call this approach global alignment, and the value of the pairwise alignment can be calculated in a similar way to that described for the multiple sequence alignment. One of the most popular pairwise alignment algorithms is the Needleman-Wunsch method [31], which is based on a dynamic programming approach. The local alignment method has a slightly different goal, because local alignments are suspected of containing regions of similarity or similar sequence motifs –motifs being short parts of sequences which are the same– within their larger sequence context. The Smith-Waterman algorithm [2] is a general local alignment method and it also based on dynamic programming. With sufficiently similar sequences, there is no real difference between local and global alignments.

```
IGRHRYHIG-G
 :   ||  ||  |
-S--RY-IGRG
```

Figure 3.1: A simple example of a Needleman-Wunsch alignment. The alignment method has identified five matches, a substitution and four deletions.

When we examine the time complexity of the above-mentioned pairwise methods we see that they both have a time complexity of $O(nm)$, where n and m are the lengths

of the sequences. With the advent of large-scale biological databases it has become necessary to speed up these methods. The Basic Local Alignment and Search Tool (BLAST) [1] is perhaps the most remarkable algorithm in this field. This method is based on an Approximate String Matching technique. Several extensions have been developed from this method like Gapped-BLAST and PSI-BLAST methods [32].

3.2 Evolutionary distances for DNA

After we have determined the places of changes/mutations in the sequences using an alignment method, we attempt to estimate the time elapsed since they diverged. There are now several approaches that have been developed for modelling the evolution of DNA sequences. Based on this models, we can infer the evolutionary relationship of the sequences. But now we shall provide an overview of these special techniques.

3.2.1 The Evolutionary Markov Process

We will commence with an introduction to the general model, because the latter models are the restricted forms of this. First we need to define the notion of a time continuous Markov chain. In our introduction we rely on the canonic textbook of Felsenstein [21] and the study by Müller and Vingron[33].

Definition 3.1 *A time continuous Markov chain is a sequence of random variables X_t, $t \in \mathcal{R}_0^+$ taking values of a finite set of states $\mathcal{A}$. X_{t_0} is distributed as $\pi^{(0)}$ and the following Markov property holds:*

$$P\left(X_{t_n} = x_n | X_{t_{n-1}} = x_{n-1}, \ldots, X_{t_0} = x_0\right) = P\left(X_{t_n} = x_n | X_{t_{n-1}} = x_{n-1}\right) \quad (3.2)$$

for any $n \in \mathbb{N}$, time points $t_0 < t_1 < ... < t_n$ and any states $x_0, x_1, \ldots, x_n \in \mathcal{A}$.

The Markov chain is time homogeneous if there exists a transition probability matrix $P(t)$ such that

$$P\left(X_{s+t} = j | X_s = i\right) = P_{ij}\left(t\right), \forall s, t \geq 0, i, j \in \mathcal{A} \quad (3.3)$$

The transition probability matrix $P(t)$ is a stochastic matrix and has the following properties:

- $P(0) = I, \quad$ *I - identity matrix*
- $P_{ij}(t) \geq 0 \quad$ *and* $\quad \sum_j P_{ij}(t) = 1,$
- $P(s+t) = P(s)P(t) \quad$ *for* $\quad s, t \geq 0.$

The time continuous Markov chain is irreducible if, for any period $t > 0$, each state can be reached from each state: $P_{ij}(t) > 0, \quad \forall i, j \in \mathcal{A}$. In this case there exists a unique stationary distribution π which is the solution of $\pi P(t) = \pi$.

If we assume that the probability transition matrix $P(t)$ is continuous and differentiable at any $t > 0$ then the following limit should exist:

$$\lim_{t \to 0} \frac{P(t) - I}{t} = Q \tag{3.4}$$

Q is known as the rate matrix or the generator of the Markov chain, and it is not hard to show that the transition can be expressed in terms of the rate matrix:

$$P(t) = e^{Qt} \tag{3.5}$$

Definition 3.2 *We call a time continuous Markov chain X_t on the set of states $\mathcal{A}$ an evolutionary Markov process (EMP) with the stationary distribution π on the states if*

1. *X_t is time homogeneous.*
2. *X_t is stationary and the initial distribution $\pi(0)$ is the stationary distribution π. Therefore X_t is distributed according to for all $t \in R_0^+$.*
3. *Xt is irreducible: $P_{ij}(t) > 0$ for all $t > 0$ and $i, j \in \mathcal{A}$.*
4. *X_t is calibrated to* 1 percent of expected mutations*: $\sum_i \pi_i Q_{ii} = -1$.*
5. *X_t is reversible: $\pi_i P_{ij}(t) = \pi_j P_{ji}(t)$ for all $t > 0$ and $i, j \in A$.*

The states of an EMP correspond to the alphabet we are using, i.e. in this case nucleic acids, so this EMP has four states. The fourth property of an EMP requires some additional comments. The Q generator matrix provides an infinitesimal description of the process. The off-diagonal elements of Q are positive real numbers, thus the diagonal elements are negative ones, and they are equal to the sum of the off-diagonal elements. Hence we can regulate the expected number of changes per time units by the calibration of the diagonal elements. This calibration step is common to all Markov process-based evolutionary models.

The relative number of changes that the pairwise alignment has revealed can express the evolutionary relationship of two sequences as well, but this EMP can also accommodate the duplicated changes. For example a position is in state 'C' and then it is substituted by 'G', and the 'G' is substituted again by 'A' in the course of evolution.

3.2.2 The General Time Reversible (GTR) model

In the modelling of DNA sequence evolution the EMP is known as the General Time Reversible (GTR) model. All of the models we will introduce later are a specialized case of this. The rate matrix of the GTR can be seen in Table 3.1. The model itself has 10 free parameters. The $(\pi_A, \pi_G, \pi_C, \pi_T)$ is the stationary distribution which is also called as the equilibrium frequencies. The off-diagonal elements are calibrated in the way described earlier. Now let us assume that we have two aligned DNA sequences s_1 and s_2 of length l. Using the EMP model we can infer the time t that has elapsed

From/To	A	G	C	T
A	-	$\pi_G\alpha$	$\pi_C\beta$	$\pi_T\gamma$
G	$\pi_A\alpha$	-	$\pi_C\delta$	$\pi_T\epsilon$
C	$\pi_A\beta$	$\pi_G\delta$	-	$\pi_T\eta$
T	$\pi_A\gamma$	$\pi_G\epsilon$	$\pi_C\eta$	-

Table 3.1: The parameters of the GTR model of DNA evolution.

since two sequences have evolved via an optimization task. Then we need to maximize the following likelihood function:

$$L(t) = \sum_{i=1}^{l} \pi_{s_1^i} P_{s_1^i, s_2^i}(t) \tag{3.6}$$

The optimal value of t can also be determined in a direct way. We will show how we can do this with a simple example borrowed from Felsenstein[21]. We can count the different types of changes between the aligned sequences. Our little example is given in Table 3.2. In this simple example we counted 500 distinct changes. The diagonal elements of the table contain the number of positions where the sequences have preserved their states. This empirical matrix is regarded as an estimation of the transversion matrix of an EMP at a particular time t, this time value being the focus of interest. But before we use this empirical matrix we need to symmetrize it and to sum up its columns to one. Then we will get an empirical matrix $\hat{P}$. Reformulating the Equation 3.5, we can estimate the value of $\hat{A}t$ via

$$\hat{A}t = \log\left(\hat{P}\right) \tag{3.7}$$

Because our EMP is calibrated, we can constrain our problem. The D matrix will be the diagonal matrix whose diagonal elements are the stationary distribution of the EMP. In this case the following equation is valid because we have assumed that the changes which occur per time unit are all equal to one:

$$\text{trace}\left(\hat{A}\hat{D}\right) = -1 \tag{3.8}$$

After, we can calculate directly the value of t:

$$\hat{t} = \text{trace}\left(\hat{A}t\hat{D}\right) \tag{3.9}$$

In this way we can infer the evolutionary distance between two sequences. Doing the calculation in our example we find that the evolutionary distance between s_1 and s_2 is $t = 0.228125$.

	A	G	C	T	Total
A	93	13	3	3	112
G	10	105	3	4	122
C	6	4	113	18	141
T	7	4	21	93	125
Total	116	126	140	118	500

Table 3.2: A simple example for the distribution of changes which have occurred in our example.

3.2.3 The Jukes-Cantor distance

The Jukes-Cantor (JC) model is the most simplest model. The JC model assumes that each type of change in a sequence occurs with the same probability. Hence the rate matrix is

$$Q = \begin{pmatrix} -3\alpha & \alpha & \alpha & \alpha \\ \alpha & -3\alpha & \alpha & \alpha \\ \alpha & \alpha & -3\alpha & \alpha \\ \alpha & \alpha & \alpha & -3\alpha \end{pmatrix}. \tag{3.10}$$

Moreover, the JC model assumes that the stationary distribution of the nucleic acids obeys a uniform distribution. So each nucleic acid occurs with the same probability in all positions. Using this model we can infer the evolutionary distance in a closed form. For two aligned sequences of length l, the number of mismatches between them will be denoted by u. The value of u/l is also known as the naive distance estimation. After some calculations we find that the Jukes-Cantor distance can be expressed by the following simple formula:

$$d_{JC}(s_1, s_2) = -\frac{3}{4}\ln\left(1 - \frac{3u}{4l}\right) \tag{3.11}$$

A very similar model was introduced in [34], but in this case the stationary frequency is not uniform.

3.2.4 The Kimura 2-parameter distance

In this section we will focus on an important model for nucleotide substitutions. This model makes a distinction between the two types of mutation in the sequence. To be precise, this means that here we can assume that the number of transitions ($A \leftrightarrow G, C \leftrightarrow T$) are more frequent than the transversion (the remaining type of changes). The rate matrix of this model is then defined as:

$$Q = \begin{pmatrix} -\alpha - 2\beta & \alpha & \beta & \beta \\ \alpha & -\alpha - 2\beta & \beta & \beta \\ \beta & \beta & -\alpha - 2\beta & \alpha \\ \beta & \beta & \alpha & -\alpha - 2\beta \end{pmatrix}. \tag{3.12}$$

With our above assumption we expect that $\alpha < \beta$ for all values of α and β. The stationary distribution is also uniform here, and because of we assumed that the EMP is calibrated, we have $\alpha + 2\beta = 1$. Here we can express the evolutionary distance in a closed form as well, just like before. First, let us assume that we have two aligned sequences, and let us denote the transition and transversion by P and Q, respectively. The Kimura-2-parameter distance for the these two sequences is then given by:

$$d = -\frac{\log(1 - 2 * P - Q)}{2} - \frac{\log(1 - 2 * Q)}{4} \tag{3.13}$$

3.2.5 The Hasegawa-Kishino-Yano distance

The Kimura 2-parameter model and Jukes-Cantor model make great restrictions on the DNA sequences because they dramatically reduce the necessary computational efforts. For example, they assume that each DNA occurs with the same probability in each position. The Hasegawa-Kishino-Yano (HKY) model is very similar to the Kimura 2-parameter model, but this estimates the stationary distribution as well. In accordance with this the rate matrix of HKY is can be written as:

$$Q = \begin{pmatrix} - & \pi_G\alpha & \pi_C\beta & \pi_T\beta \\ \pi_A\alpha & - & \pi_C\beta & \pi_T\beta \\ \pi_A\beta & \pi_G\beta & - & \pi_T\alpha \\ \pi_A\beta & \pi_G\beta & \pi_C\alpha & - \end{pmatrix}. \tag{3.14}$$

The evolutionary distance based on this model can be expressed in closed form, as well, but the description of this is beyond our scope. The interested reader can peruse [35] for the detailed derivation of this distance function.

3.3 Evolutionary distances for proteins

3.3.1 Correctional approach

Now let us consider a pairwise alignment of two sequences of amino acids, s_1 and s_2. We will denote the length of the aligned sequences by l. The number of position where the aligned sequences differ shall be denoted by u. Then we can readily define the p-distance (or naive distance) of s_1 and s_2 as

$$d_p = \frac{u}{l}. \tag{3.15}$$

This sequence distance is used to underestimate the evolutionary distance if two

proteins are closely related. That is why this distance needs to be corrected. The correctional process is based on a simple assumption that the number of amino acid substitutions at each site obeys a distribution [36]. Two widely used distributions are in use for this purpose, namely the Poisson distribution and the Gamma distribution. In Figure 3.3.1 the function of the correctional processes can be seen. Because we have assumed that the number of substitutions follows a given distribution, we need to use the inverse function of the density function.

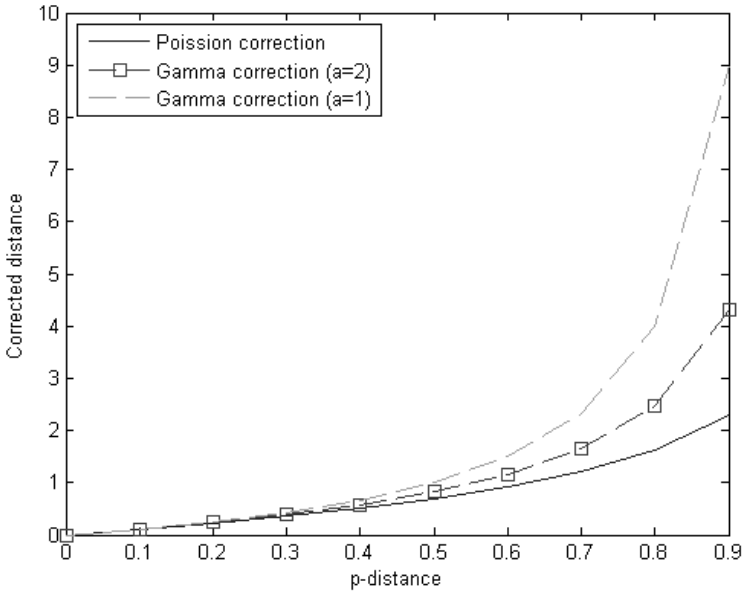


Figure 3.2: The density function of the distributions of the correctional distances. The Gamma distribution has been plotted with two different parameters ($a = 2, a = 1$)

3.3.2 Substitution models

These models specify empirical relative rates of substitution and equilibrium amino acid frequencies. Here we will focus the BLOSUM (BLocks SUbstitution Matrix) model[28]. In this probabilistic model the changes among amino acids are based on tabulating changes among closely related sequences. The closely related sequences were chosen using the FASTA program, and then it identified the highly similar region of these sequences. The changes can then be made based on this so-called highly conserved regions.

3.4 An information theoretical distance

The information theoretical sequence distances used are known as alignment-free distances because they do not require any preliminary alignment of the sequences [37]. The simplest information theoretical sequence is the relative entropy of the amino acid distribution in the sequences of interest which can also be interpreted as a sequence distance. In our experiments we applied a recently introduced distance function.

The *information theoretical distance* functions are based on a comparison of how many information sequences there are relative to each other. This approach originated from Kolmogorov-complexity theory. The Conditional Kolmogorov complexity $K(X|Y)$ is defined as the length of the shortest program computing X on an input Y [38]. The Kolmogorov complexity $K(X)$ of a sequence X is a shorthand notation for $K(X|\lambda)$, where λ is an empty sequence. The corresponding distance function uses the relative decrease in complexity or conditional complexity as a measure of sequence similarity, that is

$$d(X,Y) = \frac{max\{K(X|Y), K(Y|X)\}}{K(YX)} \tag{3.16}$$

Kolmogorov complexity is a non-computable notion, so in practical applications it is approximated by the length of a compressed sequence calculated with a compression algorithms like LZW [39] or Sequitur [40]. The formula for calculating compression-based similarity measures (CBM) using the length values of compressed sequences can be derived from Equation 3.16. It takes the form

$$d_{CBM}(X,Y) = \frac{C(XY) - min\{C(X), C(Y)\}}{max\{C(X), C(Y)\}}, \tag{3.17}$$

where $C(.)$ denotes the length of a compressed sequence, compressed by a particular compressor C. We will focus on two well-known compressor algorithm in our experiments, namely LZW [39] and Sequitur [40].

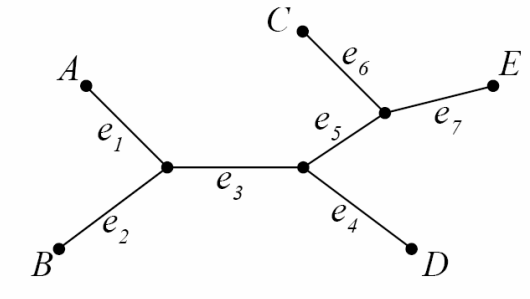
3.5 Tree criteria

3.5.1 The Least-Squares Criterion

There are many criteria in use for phylogenetic trees that can be applied to the distance data, like Minimum Evolution Length and Least Squares Criteria. These criteria usually assume that the only data items available are distances. The number of possible tree topologies grows exponentially with the cardinality of the taxon's set. Hence it is a fundamental and crucial point for the evaluation of a criterion that it should have a low time complexity for a given phylogenetic tree.

There are many forms of Least Squares criteria available, and all of them require the optimization of a quadratic function. Before we formally describe these criteria, let us denote the *path-edge incidence* or *topology matrix* of a phylogenetic tree T by P_T. The matrix P_T is a binary matrix whose columns correspond to the edges of T,while the rows correspond to the paths between the leaves of T. This representation of a tree requires a space of $O(n^3)$ because it has $n-1$ columns and $\binom{n}{2}$ rows, even though it has just a few non-zero elements. Hence it is worth exploiting the sparsity of the

topology matrix for an efficient implementation. In Figure 3.5.1 a phylogenetic tree can be seen and its encoding into a path-edge incidence matrix.



	e_1	e_2	e_3	e_4	e_5	e_6	e_7
(A,B)	1	1	0	0	0	0	0
(A,C)	1	0	1	0	1	1	0
(A,D)	1	0	1	1	0	0	0
(A,E)	1	0	1	0	1	0	1
(B,C)	0	1	1	0	1	1	0
(B,D)	0	1	1	1	0	0	0
(B,E)	0	1	1	0	1	0	1
(C,D)	0	0	0	1	1	1	0
(C,E)	0	0	0	0	0	1	1
(D,E)	0	0	0	1	1	0	1

Figure 3.3: A phylogenetic tree and its corresponding path-edge incidence matrix, where $X = \{A, B, C, D, E\}$.

Next, we shall denote a distance matrix by D defined on the taxon set of $\mathcal{T}$. We can rewrite D using its vector form $\mathbf{d}$ (i.e. turning the upper triangular of D into a vector). The arrangement according to the topology matrix P_T determines an unambiguously ordering among the $\binom{n}{2}$) entries of the vector $\mathbf{d}$. Introducing the necessary notations, we can write, in a simple way, the Unweighted Least Square Criteria (LSC) for a given T phylogenetic tree. The edge weighting of a tree T satisfies the LSC criteria if and only if it minimises the following optimisation task:

$$\min_{\mathbf{x}\in\mathbb{R}^{\mathbf{n-1}}} \| \left(P_T\mathbf{x} - \mathbf{d}\right) \|, \tag{3.18}$$

where the elements of $\mathbf{x}$ can have any real value, including zero or negative values. The solution of the problem defined by Eq. (3.18) results in an optimal edge weighting for a phylogenetic tree T and a minimal Frobenius norm for $\|D_T - D\|_F$. This means that the deviation between the calculated weighting D_T and D is minimal. The problem can be solved in $O\left(n^2\right)$ time for a given phylogenetic tree [41].

3.5.2 The Constrained Least Squares Criterion

We also require that a weighting always be positive because a negative evolution distance has no physical sense. This is why we shall restrict ourselves here to the following minimization problem:

$$\begin{aligned} \min_{\mathbf{x}\in\mathbb{R}^{\mathbf{n-1}}} & \| \left(P_T\mathbf{x} - \mathbf{d}\right) \| \\ s.t. \quad & \mathbf{0} \leq \mathbf{x} \end{aligned} \tag{3.19}$$

The Constrained Least Squares Criteria (CLSC) defined above results in a non-negative

weighting for a phylogenetic tree T. CLSC also retains the property of the original LSC that it can be solved in $O(n^2)$ time, because the algorithm introduced by Bryant& Waddell [41] can handle the Levenberg-Marquardt method as well. Here $\|D_T - D\|_F$ is the *distance estimation error (DEE)*, and

$$\frac{\|D_T - D\|_F}{\binom{n}{2}}$$

is the *normalized distance estimation error (NDEE)* of the phylogenetic tree and will be denoted by e_T.

3.5.3 The Minimum Evolution Criterion

The Minimum-Evolution (ME) criterion for phylogenetic inference is based on the assumption that the tree with the smallest sum of edge length estimates is the one most likely to be the correct one [42]. Hence with this criterion we first infer the edge length of a T tree, e.g. using a sort of least-squares criterion. After we have assessed how the tree fits to the sequenced data, we will prefer those tree whose sum of edge lengths are smaller.

3.5.4 The Maximum Parsimony Criterion

The Maximum Parsimony (MP) criterion is the most-widely used one in phylogenetic studies. Fitch first introduced this approach in his pioneering work [43].

Before we describe this criterion in more detail, let us assume that we have a T phylogenetic tree whose leaves represents sequences of length l. Then the $\mathcal{T}_T$ taxon set of this T is characters that are of fixed length. After this, we assign sequences of length l to the interior node of T. Hence at the endpoints of each of the e edges there are two sequences, s_e^1 and s_e^2. Let us evaluate the number of positions where s_e^1 and s_e^2 differ by c_e. This is equal to the changes which occurred on the edge e. The parsimony score of T is then the sum of the changes:

$$p_T = \sum_{e \in E(T)} c_e \tag{3.20}$$

The first question which arises is how we can assign the sequences to the interior point of T in such a way that the parsimony score of T will be minimal. To do this, there is a simple algorithm which is based on dynamic programming[43]. But we should mention here that finding the most parsimonious tree for a given set of sequence is NP-hard [44]. The only currently available, efficient way of obtaining a solution, given an arbitrarily large set of taxa, is by using heuristic methods but these do not guarantee that the most parsimonious tree will be recovered.

3.5.5 The Maximum Likelihood Criterion

The Maximum Likelihood (ML) criterion is in spirit similar to the Maximum Parsimony Criterion, but the cost of a change in parsimony is not a function of the edge length. In the case of MPC we simply count the changes which occurred on an edge.

The conditions are similar to those described above. First, let us assume that we have a set of aligned sequences $D = \{s_1, \ldots, s_n\}$ over Σ of length l. Moreover, we also know their phylogeny which is represented by a rooted phylogenetic tree T whose edge lengths are known. In addition, we have an evolutionary model, such as the Jukes-Cantor model, which allows us to determine the $P_{xy}(t)$ probabilities, where $x, y \in \Sigma$. This means that we will then know the probability of the different types of changes on an edge of length t.

Before we define the likelihood score of a phylogenetic tree, we need to make two basic assumptions. These are that

1. the sequences evolve independently, position by position
2. the evolutionary changes on the edges are independent

In general these assumptions are not valid in real life, but they are economizing the computation of the likelihood of a tree, as we shall shortly see.

Roughly speaking, the likelihood of a tree is $P(D|T)$. In other words, it is the probability that the set D has evolved when our hypothetical tree is T. Let us now see how we can compute this value for a given set of sequences D and for a rooted phylogenetic tree T. Using the first assumption we can reformulate this likelihood value like so:

$$L = P(D|T) = \prod_{i=1}^{l} P(s_1^i, \ldots, s_n^i|T) \tag{3.21}$$

This means that we can handle the likelihood of each position independently, thus we need just to compute the likelihood for a constrained tree, where the leaves represent just single letters. After, we can assign a letter from Σ to an interior node of T. Let us examine the case when an interior point g has been labelled by a letter $x \in \Sigma$, that is by g_x. The second assumption allows us to handle the lineages separately. Then we can compute the conditional likelihood of an interior node g if we know the conditional likelihood of its immediate descendants g^1 and g^2 via the following formula:

$$L_g(x) = \left(\sum_{y \in \Sigma} P_{g_x g_y^1}(t)\, L_{g^1}(y) \right) \left(\sum_{z \in \Sigma} P_{g_x g_z^2}(t)\, L_{g^2}(z) \right) \tag{3.22}$$

As we mentioned above, the $P_{xy}(t)$ probabilities are known. Next, we need to calculate the likelihoods of the leaves. This is simply defined as a Kronecker-delta function. For a leaf l will be defined as

$$L_l(x) = \begin{cases} 1, & \text{if leaf } l \text{ represents x} \\ 0, & \text{otherwise} \end{cases} \tag{3.23}$$

In the end, the total likelihood of the tree T is the sum of all conditional likelihoods at the root, weighted by the background discrete probability π_x of the letters

$$L_T = \sum_{x \in \Sigma} \pi_x L(r), \tag{3.24}$$

where r is the root of T. This can be computed using a dynamic programming approach [26]. But we want to calculate the maximum likelihood for a tree topology. So we need to determine the edge lengths such that the likelihood score of the phylogenetic tree T will be maximal. This can be done by applying the simple Newton-method, say. We should mention here that the likelihood function is not always convex.

Of course, the likelihood criterion can be defined in a straightforward way: we prefer those trees which have a higher likelihood value. There are many extensions for computing the likelihood score. The most remarkable was introduced by Churchill and Felsenstein [45]. In their article the first assumption was resolved because they modelled the dependencies of the neighboring positions by a Hidden Markov Model. We will also use this procedure for calculating the likelihood values in our experiments.

Chapter 4

A Tree Building Method Based On The Least-Squares Criteria

4.1 Introduction

The reliable reconstruction of a tree topology from a set of homologous, sequenced data is one of the most important goals in system biology. A major family of the phylogenetic tree building methods is the *distance-based or distance matrix methods*. The general idea behind them is to calculate a measure for the distance between each pair of taxons, and then find a tree that predicts the observed set of distances as closely as possible. There are quite a few heuristic distance-based algorithms with a fixed criterion available for estimating phylogeny, and their strengths and weaknesses are familiar to everyone in the field. The distance-based methods, like the Unweighted Pair-Group Method using Arithmetic averages (UPGMA) [10] and Neighbor-Joining (NJ) [9], work similarly: they iteratively form clusters, always choosing the best possibility based on a given criterion. We can call these methods greedy in a certain sense, because they always work on the current best candidate subtrees. The NJ method produces additive trees, while UPGMA assumes that the evolutionary process can be represented by an ultrametric tree. These restrictions may then interfere with the correct estimation of the evolutionary process. Atteson [46] showed, however, that the NJ method is able to return the true phylogeny, when the observed distance is sufficiently close to the true evolutionary distance.

The chief aim of this chapter is to develop a good distance-based method that approximates closely to the true tree for any available evolutionary (not just for ultrametric or additive) distance. To achieve this we apply a special form of the Least Square Criteria (LSC) to phylogenetic trees [43]. The LSC will guarantee a minimal deviation between the evolutionary distances and the leaf distances in the phylogenetic tree. It is fortunate that the LSC weighting for a phylogenetic tree can be computed in $O(n^2)$ time. The original LSC was introduced by Fitch and Margoliash, and nowadays several forms of it are in use in the literature, like the Weighted LSC [47], Unweighted and Generalized LSC [48]. We applied the constrained version of LSC (CLSC) here to evaluate phylogenetic trees because the weights of the edges have to be non-negative.

The solution of the problem retains its simplicity because the Constrained LSC can easily be handled by the Levenberg-Marquardt method [49].

Since finding the least squares tree (whether it is constrained or not) is an NP-complete problem [50], a polynomial-time algorithm to solve it is unlikely to exist. Many meta-heuristics have been applied in phylogenetic tree-building such as the Genetic Algorithm [51], the Tree Fusing approach [52], the Branch and Bound approach [53], the Maximum Likelihood approach [26] and the Fitch&Margoliash approach, the latter being an extension of UPGMA.

We now propose a novel heuristic, based on the so-called Multi-Stack (MS) construction [54]. The MS heuristic organizes the candidate subtrees having the same number of leaves into a priority queue according to their distance estimation error, and generates newer candidate trees by joining the existing trees via a novel tree joining strategy. It may happen however that there are many trees within a priority queue that have a non-disjunct set of leaves, and it is not possible to join them. The Closest-Neighborhood Tree Joining (CNTJ) strategy introduced here always provides a tree topology based on all of the subtrees, swapping their common taxa with their closest neighbor.

Our method was tested on artificial as well as on real-life datasets like Primates, Myoglobins and Hydrogenases.

4.2 Materials and Methods

4.2.1 Multi-Stack Approach

To solve problems which have enormous solution spaces we need to apply an efficient search technique. That is why we decided to adopt a heuristic approach for phylogenetic tree-building which is also used in speech recognition [55]. To describe the method we first have to give a definition. A *stack* is a structure for keeping candidate solutions in. In addition, we use limited-sized stacks: if there are too many candidates in a stack, we prune the ones with the highest fitness value.

In the MS algorithm we assign a separate stack to the trees having the same number of leaves and store the K-best candidate subtrees in the stack according to their DEEs. In the initial step the algorithm generates the lower level subtrees only, and then it pops each pairs of candidate subtree from the stacks, joins them in every possible way, and afterwards puts the new candidate subtrees into the stack according their leaf numbers associated with their new DEE. Applying this heuristic to phylogenetic tree building, we obtain an iterative tree-building procedure.

The pseudocode of the MS algorithm is presented in Table 4.2.1, where the Q_n elements denote the limited priority queue which contains at most K trees, and each tree has exactly n leaves in it. The initial step of the method includes the exploration of all tree topologies with at most three leaves. This step makes sense because there are only $n + \binom{n}{2} + \binom{n}{3}$ phylogenetic trees when $|\mathcal{T}| = n$, so during this initial step we can explore the whole space of trees. In the next steps MS generates the possible

subtrees, and it always keeps the best K subtrees based on their distance estimation error.

Input:	D distance matrix, K size of priority queues
1	Initial step: fill up Q_1, Q_2 and Q_3
2	for $i = 3 : n$
3	for $j = 1 : min(n - i, i)$
5	Generate all of the trees joining the elements of Q_i and Q_j
6	Add them to the priority queue Q_{i+j} according their DEE
7	endfor
8	endfor
9	return to first element of Q_n^K
Output:	T rooted phylogenetic tree with n leaves

Table 4.1: The Multi-Stack algorithm.

The complexity of the MS method naturally depends on the variable K because we are exploring an nK tree topology. Since CLSC requires a quadratic time complexity ($O(n^2)$), it becomes the most time-consuming step of the MS. Due to this features the MS tree building method has a time complexity of $O(Kn^3)$ overall.

4.2.2 Closest Neighborhood Tree Joining operator

With the Multi-Stack tree building approach it may happen that we want to join two candidate trees that have some common taxa (i.e. the set of taxa whose elements occur in the taxon set of both trees). The simplest idea is the naive approach: let us replace the common taxon set of the candidate trees that interferes the tree joining in every possible way with those taxa that do not occur in the taxon set of candidate trees. After we have carried out and evaluated all possible replacements, let us choose the best replacement. But it can be easily seen that this will lead to a very high computational burden, because the number of possible replacement grows exponentially with the cardinality of the common taxon set. Instead here we suggest a tree joining strategy as a way of avoiding this problem.

The situation is the following: we need to join two candidate subtrees T_1, T_2 having n_1 and n_2 leaves respectively, and we need to determine a strategy for the elimination of the duplicated taxa of the candidate trees: $|\mathcal{T}_{T_1} \cap \mathcal{T}_{T_2}| = n$. From the solution of this problem we also require that the distance estimation errors e_{T_1} and e_{T_2} with respect to the applied distance matrix D remain or grow as little as possible. Thus the goal here is to determine a strategy for the replacement of common taxa that produces the least variation in the tree estimation errors of the candidate trees in question.

For the formal description let us denote the cost of the replacement for a taxon by $t \in \mathcal{T}_T$ by $c(t, t')$, where $t' \in \mathcal{T} - \mathcal{T}_T$. This leads to a change in the e_T value after the replacement, which can be a negative real number as well. We can readily determine

an upper bound for this cost, because using the weights of T before the replacement, the following proposition always holds.

Lemma 4.2.1 *Let T be a phylogenetic tree with a taxon set $\mathcal{T}_T \subset \mathcal{T}$, and let e_T be its distance estimation error. A distance on $\mathcal{T}$ will be denoted by D, and the leaf distance will be denoted by D^T. Now let $t \in \mathcal{T}_T$ and $t' \in \mathcal{T} - \mathcal{T}_T$ be two taxa. Then the following inequality will hold for the $c\left(t, t'\right)$ cost of the replacement:*

$$c\left(t, t'\right) \leq \sum_{t'' \in \mathcal{T}_T} b_{t''}\left(t, t\right) - b_{t''}(t', t) \tag{4.1}$$

where $b^1_{t''}\left(t_1, t_2\right) = \left|D\left(t_1, t''\right) - D^T\left(t_2, t''\right)\right|$

Proof
If we use the CLSC for T, then we get an optimal edge weighting w using the taxon set $\mathcal{T}_T$ and distance matrix D. Equation 4.1 corresponds to the rows in Equation 3.19, that it is represents the path between t and $\mathcal{T} - t$. Thus if we replace this taxon, the change of the optima will vary according to the magnitude when we use the weighting w. So Equation 4.1 will hold apart from the choice of $t' \in \mathcal{T}$. □

Summarizing the above points, Proposition 4.2.1 allows us to determine an upper bound for a replacement of a taxon. That is why we suggest here that the common taxon set $|\mathcal{T}_{T_1} \cap \mathcal{T}_{T_2}|$ should be replaced iteratively, taxon by taxon, always choosing the pair of taxa that have the lowest bound, in accordance with Proposition 4.2.1.

4.2.3 Distances and similarities

In bioinformatics there are two classes of distances that are widely used, namely evolutionary distances and information distances [56], as we outlined previously. In this section we will describe the tools and their parameters as we used in our experiments.

Evolutionary distances.

The global alignment of protein sequences can be performed using the well-known Needlemann-Wunsch [31] algorithm with the BLOSUM70 [28] matrix. The simplest evolutionary distance between a pair of aligned sequences is usually measured by the number of sites where a substitution occurs. This measure is usually called a Hamming-distance, after assuming that the evolutionary rate is roughly constant, and that the evolutionary process is linear.

Many models have been proposed to describe the true evolutionary process. There are many corrections of this measure which try to fine tune the evolutionary rate. Some of them were used here when we performed our tests on different real-life datasets. These include the Gamma, Poisson and Jukes-Cantor corrections [57].

Table 4.2: The performance of the test on randomly generated model trees. The values in bold show the minimal value in each row.

	No leaves	Length of ancestor = 300			
		UPGMA	NJ	FM	MS
Poisson-Poisson DEE(*10^{-3})	10	61.49	25.35	24.91	**7.62** ($K = 20$)
	20	39.38	15.32	15.56	**9.32** ($K = 20$)
	30	30.43	11.97	13.21	**6.37** ($K = 40$)
	40	24.48	9.29	9.33	**5.48** ($K = 40$)
Poisson-Sequitur DEE(*10^{-3})	10	60.83	24.69	25.13	**7.39** ($K = 20$)
	20	41.10	16.62	16.58	**10.33** ($K = 20$)
	30	30.45	11.85	11.97	**6.85** ($K = 40$)
	40	24.86	9.37	9.35	**5.12** ($K = 40$)
Poisson-Poisson BSD distance	10	0.32	0.21	0.22	**0.20** ($K = 20$)
	20	0.50	0.33	0.34	**0.28** ($K = 20$)
	30	0.63	0.41	0.41	**0.32** ($K = 40$)
	40	0.73	0.48	0.48	**0.38** ($K = 40$)
Poisson-Sequitur BSD distance	10	0.53	0.30	0.31	**0.26** ($K = 20$)
	20	0.72	0.42	0.42	**0.29** ($K = 20$)
	30	0.99	0.56	0.56	**0.38** ($K = 40$)
	40	1.09	0.61	0.61	**0.40** ($K = 40$)
	No leaves	Length of ancestor = 600			
		UPGMA	NJ	FM	MS
Poisson-Poisson DEE(*10^{-3})	10	60.83	24.69	25.13	**7.39** ($K = 20$)
	20	41.10	16.62	16.58	**10.33** ($K = 20$)
	30	30.45	11.85	11.97	**6.85** ($K = 40$)
	40	24.86	9.37	9.35	**5.12** ($K = 40$)
Poisson-Sequitur DEE(*10^{-3})	10	122.55	34.79	35.90	**17.49** ($K = 20$)
	20	70.58	16.58	16.32	**11.05** ($K = 20$)
	30	48.45	10.31	10.39	**6.61** ($K = 40$)
	40	37.32	6.16	6.06	**5.50** ($K = 40$)
Poisson-Poisson BSD distance	10	0.32	0.21	0.22	**0.20** ($K = 20$)
	20	0.50	0.33	0.34	**0.28** ($K = 20$)
	30	0.63	0.41	0.41	**0.32** ($K = 40$)
	40	0.73	0.48	0.48	**0.38** ($K = 40$)
Poisson-Sequitur BSD distance	10	0.49	0.27	0.29	**0.29** ($K = 20$)
	20	0.79	0.44	0.44	**0.32** ($K = 20$)
	30	0.95	0.54	0.54	**0.35** ($K = 40$)
	40	1.15	0.63	0.62	**0.39** ($K = 40$)

4.2.4 Generation of model populations

Since the correct phylogeny for a set of taxa is usually unknown, we first carried out our tests on randomly generated model populations having $10-20-30-40$ members. For each population 100 independent and identically-distributed model trees were generated from the tree-space.

In order to calculate the leaves of these trees, pseudo random sequences of 300 and 600 amino acids were used as ancestor sequences. The sequence was then assumed to evolve according to the predetermined branching pattern of the randomly generated model tree. The edge lengths of the generated tree correspond to the expected number of amino acid substitutions per site. We varied this value between $0-0.1$, and the number of amino acid substitutions at each site was assumed to have a Poisson distribution. Two well-known substitution models, the Jukes-Cantor [57] and the Poisson [36; 58], were also used to mimic the mutations. In this way 100 different set of sequences (model populations) were generated for each (10-20-30-40) member number.

4.2.5 Description of real-life datasets

We utilized three different datasets of various size to compare and test the methods. Primates consist of mitochondrial DNA, while hydrogenases and myoglobins are distinct protein families, hence they are very suitable objects for statistically testing different tree building and distance (similarity) calculating procedures.

The set of *primates* is quite small (12 sequences), and it was borrowed from Ovchinnikov et al. [59]. This dataset contains the mitochondrial DNA of two Neanderthals, the modern human species and other vertebrates.

The second set we used for testing is an arbitrary set of sequences of *myoglobins*. It contains 27 proteins from different organisms.

The third set is the group of 75 [NiFe] *hydrogenases*. Hydrogenases are metalloenzymes that catalyze the reaction $H_2 \rightleftharpoons 2H^+ + 2e^-$. They can be found in bacteria, archae and cyanobacteria. The [Ni-Fe] hydrogenases are usually placed into 4 different taxonomic groups [60].

In the rest of the chapter these datasets will be called *primates, myoglobins and hydrogenases* respectively.

4.3 Experiments

The efficiency of our methodology was tested on model populations as well as on real-life datasets.

4.3.1 Evaluation of the model populations

The evolutionary distances (Poisson distance and CBM similarity measure with the Sequitur compressor method [40]) were calculated for the model populations, and

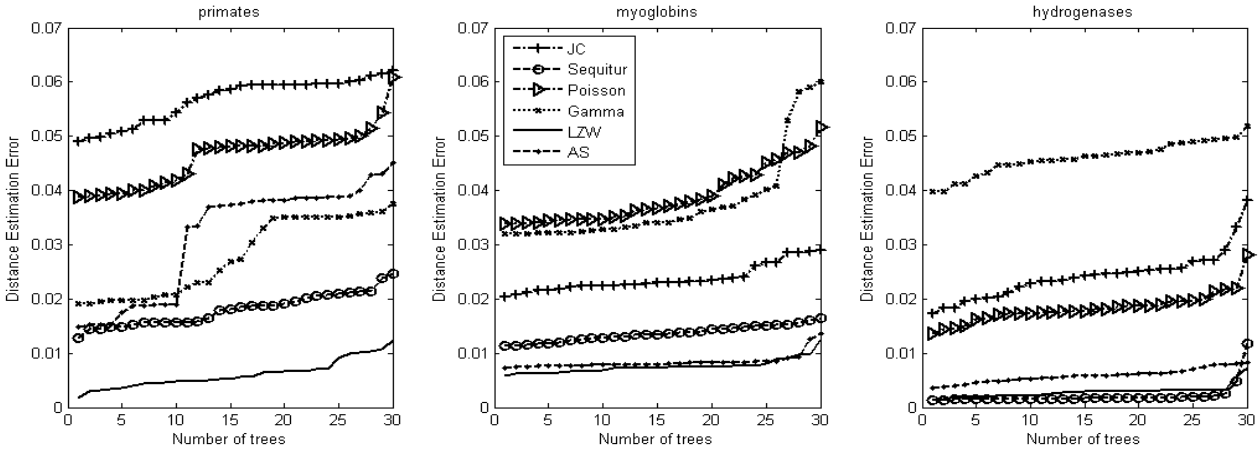


Figure 4.1: The normalized DEE of the MS trees in the last priority queue ($K = 30$).

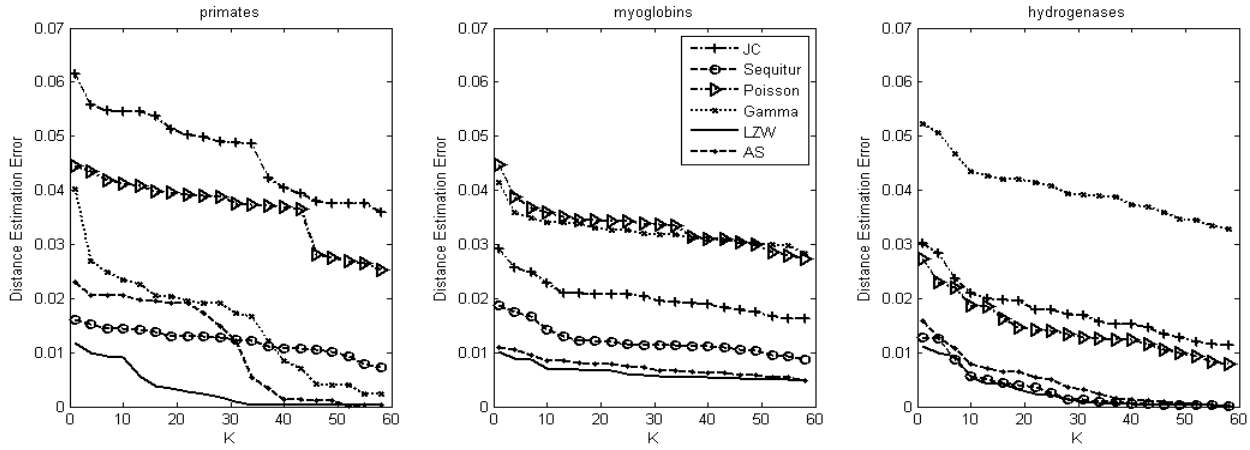


Figure 4.2: The dependence of the normalized DEE of the best tree in the priority queue on the parameter K.

phylogenetic trees were built over these model populations using four different tree-building methods: Unweighted Pair Group Method with Arithmetic Mean (UPGMA) [10], Neighbour-Joining (NJ) [9] and the Fitch-Margoliash (FM) [43] method, all of which were implemented in the Phylip package [61], and our newly developed Multi-Stack method (MS). The parameter K for the MS method was set to 20 for the populations having 10 and 20 members (leaves) and to 40 for the populations with 30 and 40 members (leaves).

The BSD distance between the randomly-generated model tree and the built phylogenetic tree along with the distance estimation error (DEE) were calculated after building the phylogenetic tree. The test was repeated 100 times on 100 similar model populations and the average of BSD distance and DEE was calculated. The results of this are summarized in Table 4.2. It is striking that the MS method is superior over all other methods tested. Moreover, both the BSD and the DEE values are smaller in every case when the MS method was applied. The UPGMA approach in contrast proved to be the least efficient method in reconstructing phylogenetic trees. The performances of the NJ and the FM methods are quite similar, and the means of the BSD distance are

equal to each other in many test cases. The mean of the Normalized DEE and BSD distances for NJ and FM only lags behind the results of the MS method by a small amount when the leaf number is set to 40.

4.3.2 Real-life Datasets

The newly developed MS method was tested on real-life datasets as well. To evaluate the trees we used the distance estimation error values (DEE). The properties of the MS method were investigated and the results were again compared with other tree building algorithms (The Unweighted Pair Group Method with Arithmetic Mean (UPGMA), Neighbour-Joining (NJ) and the Fitch-Margoliash methods implemented in the Phylip package). In this case we applied them on six evolutionary distances (similarities) (Jukes-Cantor, Gamma, Poisson, LZW, Sequitur and alignment score).

The only tunable parameter for the MS method is K (the size of the limited priority queue). It is a crucial question of how big or small this value should be in order to get reliable trees. When evaluating our MS trees we always chose the best tree in the last stack, i.e. the one which had the lowest DEE value. It is interesting to see the "goodness" of different trees in the last stack i.e. how much the "best tree" was better than the others. We set the value of parameter K to 30 and plotted the DEE value of the trees in the last stack (Figure 4.1) for primates, myoglobins and hydrogenases. The DEE for the trees grew slightly at the beginning of the stack, but the first few trees were almost just as good. There was a pronounced jump after this nearly constant level. The position of the jump depends on the evolutionary distance used, but correlates with the number of leaves on the tree when the number of leaves is small (< 30). For hydrogenases the jump was set at around $K = 30$.

In order to investigate the effect of the K parameter on the "goodness" of trees we also built trees for each dataset using different K values. The DEE value of the best tree (which has the smallest DEE) in the last queue was plotted against the K in Figure 4.2 for different phylogenetic distances and datasets. As can be seen, the DEE decreased while the limits of the priority queues rose to 60. Moreover there is threshold (about $30 - 40$), after which the DEE of the best trees remains practically constant.

As a rule of thumb these points give us a good estimation of what the parameter setting for K should be. According to this rule, K should be around the number of leaves if it is smaller than 30. For bigger trees $K = 40$ seems to be a good estimate.

Applying this rule we built trees with different tree building methods using various distances on the three real-life datasets. The results are summarized in Table 4.3. It is evident that DEE in most cases is the smallest for our new MS based tree building method. The performance of the UPGMA was not as good as the others when compared with the model populations, but the Normalized DEE for the FM and NJ methods are very similar here. Interestingly these two methods (NJ, FM) outperform the MS method in terms of a Normalized DEE when we used alignment-based evolutionary distances. Otherwise the MS method achieved better results. We also compared the methods in terms of their BSD values. In Figure 4.3 the labels along the axis represent the tree

Table 4.3: The normalized distance estimation error of different tree building methods using distinct similarity measures on the datasets. The values in bold show the minimal value in each row.

	Primates ($N = 12$)			
	UPGMA	NJ	FM	MS
JC	96.22	60.42	49.93	**45.17**
Gamma	112.87	76.36	63.08	**17.62**
Poisson	93.85	53.99	47.97	**37.06**
LZW	68.95	12.31	12.31	**1.68**
Sequitur	30.49	30.49	30.49	**11.89**
Alignment-Score	88.25	78.32	65.59	**13.71**
	Myoglobins ($N = 27$)			
	UPGMA	NJ	FM	MS
JC	88.35	62.77	62.12	**19.70**
Gamma	174.56	90.69	81.43	**30.80**
Poisson	115.63	57.34	59.53	**32.58**
LZW	**1.68**	33.10	8.63	62.97
Sequitur	69.20	23.10	48.76	**10.99**
Alignment-Score	65.66	18.21	48.05	**7.03**
	Hydrogenases ($N = 75$)			
	UPGMA	NJ	FM	MS
JC	40.80	11.69	**10.58**	15.17
Gamma	56.44	18.29	**16.97**	37.48
Poisson	37.86	10.33	**8.76**	12.36
LZW	15.33	2.91	1.85	**0.50**
Sequitur	22.86	2.13	2.14	**0.52**
Alignment-Score	26.71	4.07	5.32	**1.27**

Normalized distance estimation errors were multiplied by 1000. The value K for MS was set to 30 for primates and Myoglobins and 40 for hydrogenases.

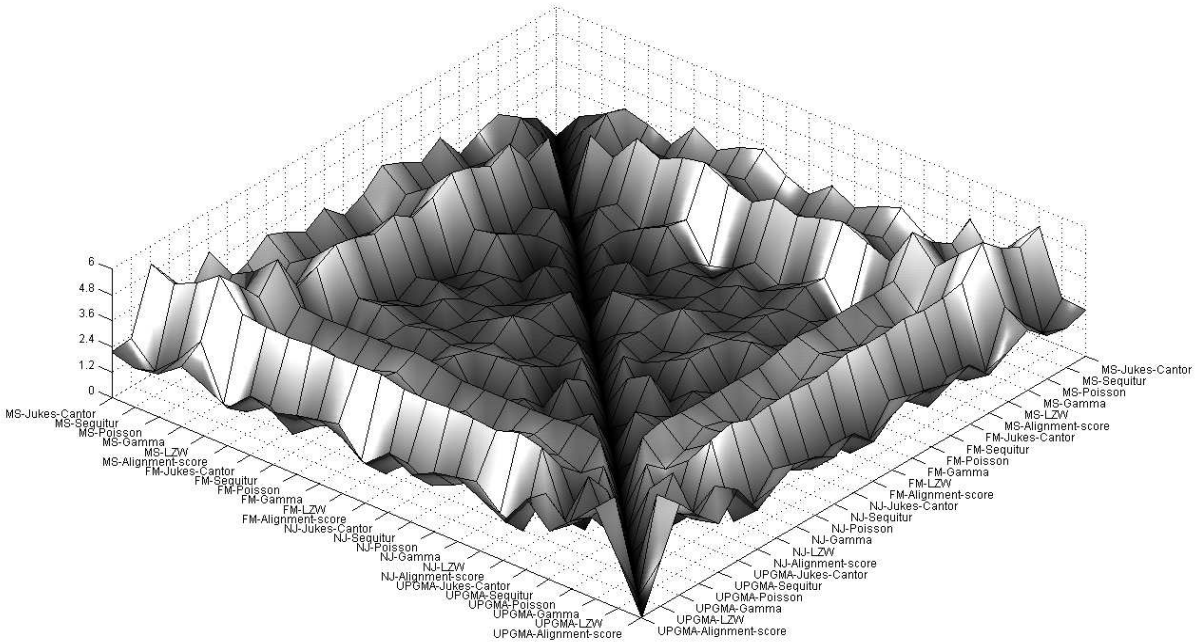


Figure 4.3: The BSD distance of the trees with the myoglobin dataset.

building methods and the evolutionary distance/similarity measure we applied, and are separated by a hyphen. Comparing the tree topologies of different trees that employ the BSD, we see that the performance of the MS method is very similar for all datasets (note the plateau in Figure 4.3). It is also apparent from the evaluations that distance-based methods (UPGMA, NJ and FM) produce trees with very similar topologies (note the wide valley in the middle of the diagrams). The topology of the trees built by the MS method are different for these trees (notice the higher regions of the diagram in Figure 4.3). The MS method, however, produced similar topologies regardless of the evolution distance used for tree building, but we can still say that the MS trees brought an improvement in the Normalized DEE for the trees as Table 4.3 quite clearly indicates.

4.4 Conclusions

In this chapter we have presented a novel distance-based and iterative tree building algorithm for analysing the lineage of taxa in structural biology and then compared it with other tree building methods using a new phylogenetic benchmark. Next, we showed for simulated model datasets and for three distinct real-life datasets, that it is an efficient tool for building phylogenetic trees. The new method is superior on distance estimation and produces robust trees as the tests on model trees show. The "goodness" of the resultant trees, however, strongly depends on the parameter K. As it follows from the nature of the method, if K is big enough the MS method approximates the exhaustive search. On choosing a proper K value, MS successfully and quickly

searches in a previously unexplored region of the possible tree topologies, hence it produces slightly different topologies than those with the algorithms used previously. This allows us to gain a deeper insight into protein and DNA evolution, relationships and lineage, and we hope that the MS method and the phylogenetic benchmarking we introduced here will become widely used tools for tackling phylogenetic problems.

Chapter 5

Consensus methods

5.1 Introduction

In biological evolutionary studies it is a commonly used technique to represent a collection of phylogenetic trees by a single tree. The algorithms used for this are the consensus tree methods. The first consensus tree method was proposed by Adams [13], and since then many methodologies have been introduced which seek to provide a 'representative' tree for a given set of phylogenetic trees. Currently the most popular methods are the strict consensus and the majority consensus methods, which are included in the Phylip [61] and the Paup [20] program packages.

Each edge of a rooted phylogenetic tree over a set of object X corresponds to a cluster of X. Hence the consensus methods work on a set of clusters C which are obtained from the rooted phylogenetic input trees, or in other words from the input profile. If set of subsets of X are compatible then we can represent by using a single (not necessarily binary) tree. Hence one should try to find the largest number of compatible clusters. The problem of finding the maximum compatible subset is NP-complete [62; 63].

In this study we will focus on a special case of this problem, where a non-negative real-valued weight function is defined on the input cluster set. We will look for a compatible subset C' of C for which the sum of the cluster weights is maximal. This particular problem is known as the Max Clique Consensus (MCC) problem, and it is also NP-complete when the number of input trees are more than two.

There are many ways one can to define a weighting function in the case of the MCC. In the simplest case, the number of trees they contain the cluster from the input tree set is assigned to this cluster as its weight. It is also possible to use the likelihood score of the input trees as a weighting, as well.

Here we present a binary integer programming formulation and solve it using the well-known Branch and Bound algorithm [64], which is a general method for finding the optimal solution of a combinatorial optimization problem. Our new approach was tested using the input trees obtained by a parsimony method of the PAUP Program Package [20] (this widely-used phylogenetic analysis method produces more than one

output trees for the input dataset).

We have compared the consensus trees calculated with our new method to those calculated by reference methods using various dataset sizes and various evolutionary models [26; 35].

The experiments clearly show that the consensus tree building algorithm presented here enhances the performance of the tree-building methods, and it has a moderate time consumption (proportional to the phylogenetic tree construction itself), so it could be employed efficiently in a post-processing phase of a phylogenetic analysis tool.

5.2 Methods

5.2.1 Consensus Tree Methods

We applied three well-known consensus methods for comparison, namely: *majority rule, strict consensus and greedy consensus*. All of these methods determine, though somehow differently, a subset of the cluster sets of the input phylogenetic trees $\mathcal{T} = \{T_1, T_2, \dots T_K\}$. The set of trees is also called as a *profile*. For a comprehensive description of different consensus tree methods, see the review of Bryant [15].

The *strict consensus* procedure simply collects those clusters (C_S) which are common to all input phylogenetic trees. That is,

$$C_S = \{C : C \in T_i^{\mathcal{C}} \quad \text{for all} \quad i \in \{1, \dots, K\}\} \tag{5.1}$$

It is obvious that if the input trees are very diverse, the strict consensus tree will be very sparse, consequently the consensus tree will not provide a good representation of the input.

The *majority consensus* method is based-on a majority voting rule. Let $N(C, \mathcal{T})$ denote the number of input trees where C can be found. Next, let us consider the following cluster set:

$$C_M^{\rho} = \{C : \frac{N(C, \mathcal{T})}{K} \geq \rho\} \tag{5.2}$$

This rule means that we only choose those clusters which can be found in the majority of the input trees. Two interesting remarks should be mentioned. First, if $\rho = 1$ then this approach is equivalent to the strict consensus algorithm, the *majority rule* tree always being a refinement of the strict consensus tree. Second, it is a simple corollary of the *Splits-Equivalence Theorem*, when $\rho \geq 0.5$, that the majority consensus tree is always a phylogenetic X-tree [22]. In our tests we always used $\rho = 0.5$, because this value is the most commonly used one in the literature.

The third algorithm we used is the greedy approach. The *greedy consensus algorithm* consists of the following two steps: first, we sort the clusters in descending order

according to their frequencies (i.e. the number of trees they appeared in). Second, we iteratively add the cluster with the highest frequency to our consensus cluster set if it obeys the restriction of being compatible with all previously added clusters. Theoretically, the greedy consensus tree is a refinement of the majority rule tree, since we can stop adding clusters when they occur in less than the half of the input trees.

5.2.2 A Binary Integer Programming Formulation of the Max Clique Consensus

Clusters are common starting points in phylogenetics, because a rooted phylogenetic tree can be determined uniquely by its cluster set. Thus one way of reconstructing a tree is to retrieve compatible clusters based on the input data. The input data can also be sequenced data, but here we will focus on consensus trees only, so the source of clustered data will be a set of rooted binary phylogenetic trees. Proceeding now with this approach, let us denote the set of input trees or profile by $\mathcal{T} = \{T_1, T_2, \ldots, T_K\}$ and their cluster set by $\mathcal{C} = \bigcup_{1 \leq i \leq K} T_i^C$. Furthermore, let us assume that there exists an associated non-negative real valued weighting function $w : \mathcal{C} \to \mathbb{R}^+$. Our goal is to find the rooted phylogenetic tree T_{MC} for which the following two properties hold:

i.) $T_{MC}^C \subseteq \mathcal{C}$

ii.) $\sum_{c \in T_{MC}^C} w(c) \to \max$

This problem is the so-called *Max Clique Consensus* problem and it is NP-complete if the number of input trees (K) is three or more [62]. In the case of $K = 2$, David Bryant showed that it can be solved in polynomial time, because the problem is equivalent to the searching for the maximum weight independent set in a bipartite graph, which can be solved in polynomial time using the "Hungarian" method [63; 65].

After this formal definition of the MCC, let us consider its formulation. Let $x_i^b \in \{0, 1\}$ denote binary variables, where $1 \leq i \leq m(= |\mathcal{C}|)$, and $x^b = (x_1^b, \ldots, x_m^b)$. The value of an x_i^b tells us whether the ith cluster has been chosen or not, when building the consensus tree. The objective function to be maximized can be written in a simple way, namely $\sum_{1 \leq i \leq m} x_i^b w(c_i)$. The compatibility of the clusters can be forced via constraints. Now let L denote the index pairs of incompatible clusters. That is,

$$L = \{(i, j) : c_i \cap c_j \neq \emptyset \quad \text{and} \quad c_i \backslash c_j \neq \emptyset \quad \text{and} \quad c_j \backslash c_i \neq \emptyset\} \tag{5.3}$$

.

None of these cluster pairs can occur together in the Max Clique Consensus tree. For a given (i, j) index pair let us consider a binary vector $a_{(i,j)}$ having all zero elements but the ith and jth components are equal to one. Then it is straightforward to formulate an incompatibility condition for these clusters via the $a_{(i,j)} vector$:

$$a_{(i,j)}^T x^b \leq 1 \tag{5.4}$$

Doing this, we will get a binary integer linear programming problem that may have an enormous number of constraints. It is still possible to handle this problem without much difficulty if we exploit the sparsity property of the constraints (there are only two non-zero elements in each constraint). There are a number of optimization toolboxes i.e. in MATLAB [66] which can handle this kind of problem. In our implementation we used the MOSEK toolbox [67], which turned out to be quite efficient at solving this sort of integer programming task. All of these packages are based on a Branch-and-Bound (BB) procedure. This approach will fairly intelligently enumerate all the possible integer solutions.

As regards the optimization problem, each constraint corresponds to an incompatible cluster pair from two distinct phylogenetic trees. We can represent these relations among the clusters using a so-called *incompatibility graph* $\mathcal{I} = (C, E)$, whose nodes correspond to the clusters of the input trees and their edges represent the incompatibilities of the clusters (i.e. there is an edge between two nodes which represent two incompatible clusters). This construction can be determined in $o(K^2n^2)$ time, where K is the size of the input profile and n is the size of the common taxon set [62]. This remark helps us to lead to an efficient implementation of the optimization problem.

If we restrict the Max Clique Consensus problem for *d-trees* (i.e. in the input and output trees every internal vertex has a degree at most d), then there is an algorithm which has an $O(n^2dK^{d-1})$ time complexity where K is the number of the clusters obtained from the input trees. This special case of MCC is called as *Maximum Weighted Consensus d-tree* and the detailed description of the algorithm can be found in [63].

5.2.3 Weighting Schemes

The associated weighting function w can be defined in many ways. We can, for instance, simply assign the frequencies ($N(C, \mathcal{T})$) of the clusters as weights. We suggest, however, a more sophisticated, likelihood-based weighting function which fits to this problem better. In order to generate trees for testing we have to use a high throughput generation of trees. The maximum parsimony-based tree building method was chosen, where each tree represents an equally possible estimate of the real tree. It is straightforward to apply the Max Clique Consensus with the likelihood-based weighting to get a refinement of the output of the tree building method.

In [68] a maximum likelihood-based weighting was applied using majority consensus, but their approach was focused on the regulation of the multiplication of input trees. Briefly, a tree was cloned in the input set when it had a higher likelihood score than the others.

Here we suggest a similar weighting scheme for the clusters of the trees, based also on the likelihood scores of the input trees. Let us assume that we have the $s_1, \ldots, s_N$

likelihood values of the input trees. Then the weight of a cluster is simply given by:

$$w(c) = \frac{\sum_{c \in T_i} s_i}{\sum_{1 \leq j \leq N} s_j}. \tag{5.5}$$

Hence the weight of a cluster is proportional to the sum of the likelihood values of the trees containing that particular cluster.

5.3 Experiments

5.3.1 Model datasets

Since the correct phylogeny for a set of taxa is usually unknown, we first carried out tests on randomly generated model populations that have a variety of members. For each population 100 independent, identically-distributed and non-ultrametric model trees were generated from the tree-space. In order to calculate the leaves of these trees, pseudo random sequences of 900 nucleic acids were used as ancestor sequences. The sequences were then assumed to evolve according to the predetermined branching pattern of a particular randomly generated model tree which assumes the Yule-Harding speciation process [69; 70]. (This process has many attractive features. For example, it is more likely to produce balanced trees, so generating a caterpillar tree has a higher probability than using a uniform distribution in the tree space. The edge lengths of a generated tree were calculated according to a standard exponential distribution with various λ parameter values. λ was varied between $0.1 - 1.0$, so the molecular clock assumption [58] was hardly deviated in our experiments, as it is found in most the biological datasets.

For the generation of the sequenced data we used the Seq-Gen program [71], which can simulate wide range of evolutionary processes. We used the Kimura 2-parameter model (K2P) [72] which assumes that the transversion and transition ratio is equal to 0.5. Furthermore, we also tested our consensus method under the evolutionary process introduced by Felsenstein (F84) [26], which is similar to the Jukes-Cantor model [57] in the sense that the transversion and transition ratio is equal, but the base frequencies are different.

5.3.2 Tree reconstruction methods and their settings

The input trees for the consensus methods were generated using the PAUP (version $4.0b10$) program. The PAUP package was used with the following settings, appended to the end of the NEXUS sequence file:

set crit=pars; hsearch start=stepwise addseq=simple swap=tbr
retain=yes nbest=x;

where the nbest parameter determines how many output trees will be generated by the PAUP program. The default values were used for the other parameters. We also tried

to evaluate the PAUP tree building method using maximum likelihood criteria, but a single tree-building time was over 30 minutes for a dataset that contains 50 sequences, so we could not finish our calculations.

The log likelihood values of the trees were estimated using Felsenstein-Churchill model [45], applying the DNAML program from the PHYLIP program package. This model assumes that the transition and transversion ratios vary from site to site corresponding to a Hidden Markov Model [73].

Table 5.1: The average of the RF differences for the model trees. The number of leaves was chosen to be $n = 50$. The λ parameter of the standard exponential distribution for which the lengths of the edges obeys, was set to $0.1 - 0.5 - 1.0$. In these experiments we used the PAUP package as we described in Section 5.3.2.

	Number of	Length of ancestor = 900				
	input trees	ORIG	MCC	GREEDY	STRICT	MAJ.
K2P $\lambda = 0.1$	10	8.63	**7.33**	8.57	12.14	8.57
	25		**7.50**	8.57	18.71	9.70
	50		**8.46**	8.48	28.62	8.47
F84 $\lambda = 0.1$	10	8.75	**7.73**	8.69	12.15	8.67
	25		**8.29**	8.57	18.96	8.59
	50		**8.53**	8.50	28.52	8.50
K2P $\lambda = 0.5$	10	15.61	**9.75**	13.54	19.63	13.46
	25		**9.78**	13.30	25.07	13.32
	50		**10.92**	13.23	29.95	13.20
F84 $\lambda = 0.5$	10	15.86	**8.87**	13.43	19.57	13.41
	25		**11.00**	13.14	24.82	13.15
	50		**11.11**	12.86	29.69	12.83
K2P $\lambda = 1.0$	10	39.23	37.59	38.06	**32.71**	37.92
	25		36.09	38.04	**33.39**	38.08
	50		37.33	37.89	**34.70**	37.91
F84 $\lambda = 1.0$	10	40.47	38.26	38.94	**32.62**	39.00
	25		38.63	38.85	**33.58**	38.91
	50		39.70	39.29	**34.85**	39.29

5.3.3 Evaluation of performance

We evaluated the consensus tree building methods in terms of RF differences described in Section 2.1. We tried to cover all types of datasets which may occur in real life. The size of the databases was limited by the computational burden of the tree reconstruction methods. Therefore, the sizes of the model datasets were set to $50 - 100 - 200$. The branch length of the trees was generated according to an standard exponential distribution. The λ parameter was set to one of three different values: $0.1 - 0.5 - 1.0$. With a 1.0 settings the model dataset had a rather high divergence. In addition, we tested the methods using two different evolutionary models: the Kimura-2-parameter (K2P) and Felsenstein84 (F84) models.

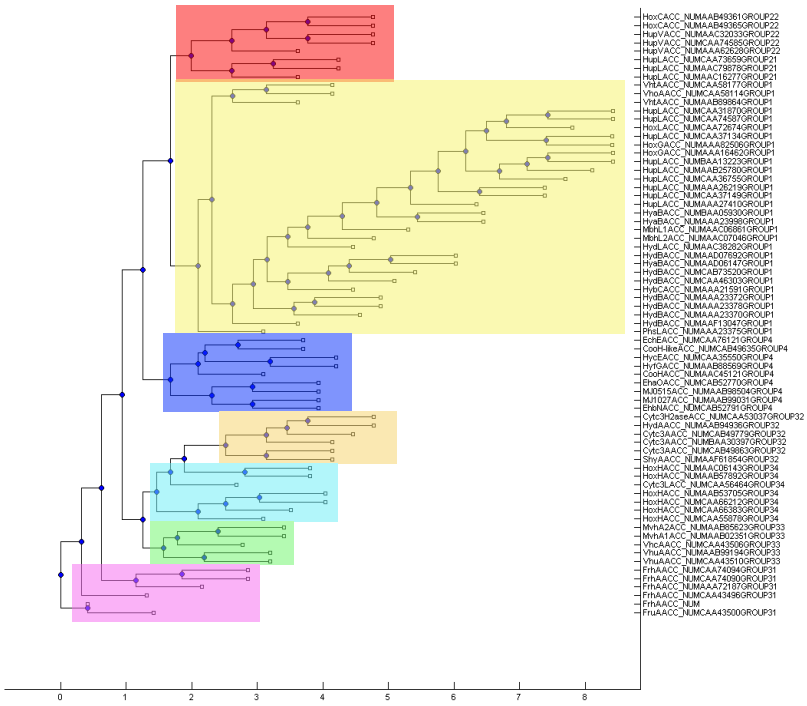


Figure 5.1: The MCC consensus tree of the hydrogenase group. The taxonomic groups themselves can be seen separately on the right side of the picture.

Table 5.1 lists the results for $n = 50$ leaves. The results reveal a few general trends. First, in each case it is worth applying a consensus method; on the other hand among the consensus methods the MCC approach using the maximum likelihood based weighting (described in Section 5.2.3) outperforms all the other algorithms in terms of symmetric difference, except when the dataset displays a high divergence (i.e. $\lambda = 1.0$). The differences between the performances are only marginally influenced by the choice of the number of input trees. The choice of the λ parameter, however, hardly influences the performance. We should mention here that if the task is very complicated (e.g. with K2P and F84 when $\lambda = 0.1$), i.e. the tree reconstruction is not so accurate, the consensus tree methods will not achieve any significant improvement on the results. In this case only the strict consensus method produces a noticeable improvement, because it generates trees which are not so resolved (i.e. their cluster set contain only a few clusters). It indicates that the information content of the strict consensus trees are very low (i.e. highly unresolved), this means that the input profile contains very different trees.

The strict consensus method achieves a higher symmetric difference, however, when the task is not so complicated. Table 5.2 shows the results for datasets of 100 sequences (the results for the dataset of 200 sequences are not shown). The general trends are quite similar to those mentioned above.

In Table 5.3 the results on amino acid sequences are presented. The general trend is very similar to the results of previous two experiments, but in this case the MCC outperforms the other consensus tree methods even in the case of $\lambda = 1$.

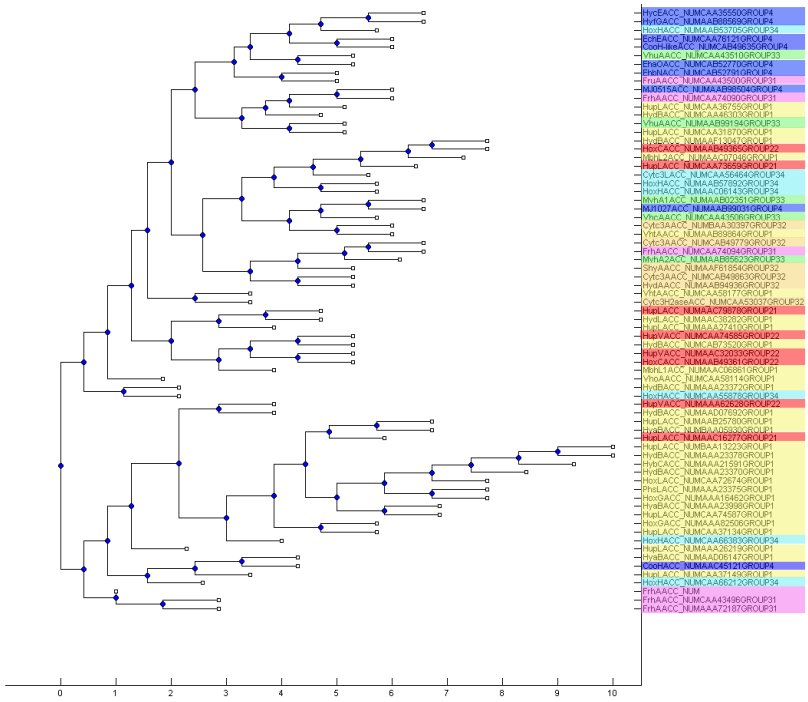


Figure 5.2: The Majority consensus tree of the hydrogenase group.

Table 5.2: The average of the RF differences for the model trees. Here the number of leaves is $n = 100$. The λ parameter of the standard exponential distribution for which the lengths of the edges obey, is set to $0.1 - 0.5 - 1.0$. In this experiment we used the PAUP package described in Section 5.3.2.

	Number of input trees	Length of ancestor = 900				
		ORIG	MCC	GREEDY	STRICT	MAJ.
K2P $\lambda = 0.1$	10	12.55	**9.51**	11.94	17.47	11.95
	25		**11.39**	11.97	24.56	12.00
	50		**11.77**	11.85	32.62	11.84
F84 $\lambda = 0.1$	10	12.16	**9.78**	11.38	16.84	11.44
	25		**11.14**	11.50	24.13	11.52
	50		**11.17**	11.26	32.32	11.28
K2P $\lambda = 0.5$	10	24.05	**15.16**	21.57	28.30	21.51
	25		**17.02**	21.48	33.96	21.43
	50		**18.88**	21.48	39.35	21.49
F84 $\lambda = 0.5$	10	23.65	**14.33**	21.33	27.96	21.27
	25		**14.32**	21.29	34.40	21.31
	50		**17.94**	21.21	39.86	21.19
K2P $\lambda = 1.0$	10	87.80	86.24	87.12	**72.01**	87.14
	25		89.98	87.23	**69.49**	87.19
	50		87.87	87.21	**68.37**	87.24
F84 $\lambda = 1.0$	10	87.05	82.75	86.57	**71.41**	86.45
	25		86.60	86.43	**68.65**	86.36
	50		74.77	86.41	**68.02**	86.39

Table 5.3: The average of the RF differences for the model trees. The ancestral sequence was an amino acid sequence with length of 500 in this test. The number of leaves was chosen $n = 100$. The λ parameter of the standard exponential distribution for which the lengths of the edges obey, was set to $0.1 - 0.5 - 1.0$. In this experiment we used the PAUP package described in Section 5.3.2.

	Number of	Length of ancestor = 500				
	input trees	ORIG	MCC	GREEDY	STRICT	MAJ.
BLOSUM $\lambda = 0.1$	10	9.74	**8.56**	9.78	12.74	9.82
	25		**8.88**	9.70	18.84	9.68
	50		**9.40**	9.70	26.74	9.68
BLOSUM $\lambda = 0.5$	10	12.27	**10.08**	11.86	15.75	11.71
	25		**10.72**	11.69	21.88	11.69
	50		**10.84**	11.42	29.55	11.36
BLOSUM $\lambda = 1.0$	10	18.87	**10.95**	16.36	21.64	16.34
	25		**13.60**	16.11	26.15	16.13
	50		**15.01**	16.04	29.94	16.09

In our experiments the computational time requirements for all four evaluated consensus methods were no more than few seconds in each test case, which is negligible when we consider the time requirements for the tree building phase.

5.3.4 A real-life dataset

The real-life dataset we used is the group of 71 [NiFe] *hydrogenases*. Hydrogenases are metallo-enzymes that catalyze the reaction $H_2 \rightleftharpoons 2H^+ + 2e^-$. They can be found in bacteria, archae and cyanobacteria. The [Ni-Fe] hydrogenases are usually placed into 4 different taxonomic groups [60].

The multiple alignment of the sequences was performed by the ClustalW program [30] using the Neighbor-Joining tree as the guide tree and the BLOSUM80 matrix [28] as the scoring matrix. The aligned sequences are available at the supplementary web site. In our experiments we employed the following tree building methods:

1. Single Linkage
2. Neighbor-Joining [9]
3. PAUP using Maximum Likelihood criteria [20]
4. PAUP using Maximum Parsimony criteria [20]
5. Multi-Stack [11]
6. Tree Puzzle [74]
7. MrBayes [75]

Here trees for both small and large subunits of hydrogenase were used for consensus determination. Altogether the profile contained 14 trees. The strict consensus algorithm resulted in a fully unresolved tree. The majority consensus and the greedy consensus tree were the same in this experiment and it is presented in Figure 9.3. The taxonomic group ID is shown after the GROUP keyword in the name of the proteins. The grouping of hydrogenases resulted from these consensus methods is confused. The previously determined taxonomic groups were badly done. In contrast the MCC approach resulted in a tree which classifies the taxonomic groups perfectly. Every single enzyme is correctly placed within its correct taxonomic group justifying the classification of the hydrogenases which was based not just on phylogenetic but also on structural and functional similarities of different enzymes [60].

5.4 Conclusions

Here we developed a binary integer programming formulation for the Max Clique Consensus problem, which is known to be an NP-complete problem. Due to this formulation the Max Clique Problem has become applicable for consensus tree building, and it outperforms the other, widely used and cited consensus tree building methods, when we use a suitable maximum-likelihood based weighting scheme. We compared the various consensus methods in the case of two evolutionary models (K2P,F84) using a well-known tree-building program package (PAUP). The MCC tree in most of the test cases is the most accurate one in terms of symmetric difference (better even than the best tree based on the parsimony score).

When the dataset shows an extremely high divergence, however, the strict consensus achieves the best performance. This superiority of the strict consensus is, however, virtual and we should rather consider it as an indication of the inaccuracy of the tree reconstruction than a real result.

We have also evaluated our method using a real-life dataset. It contained 71 [NiFe]*hydrogenases*. We have built phylogenetic trees using various tree reconstruction methods for both the small and the large subunits of the [NiFe] *hydrogenases*. The investigated consensus tree methods do not show the four taxonomic group in this protein family, while our MCC consensus method achieved the right taxonomic classification, in agreement with the generally accepted phylogeny.

It is obvious from the data that the MCC consensus outperforms many widely-used procedures, and it is easy to implement. The time requirement of this method is reasonable (proportional to the tree building method itself), and the MATLAB implementation of this is accessible freely from the supplementary web page. That is why we hope that it will be a widely used tool in the area of phylogenetic tree reconstruction.

Part II

Phylogenomics

Chapter 6

Introduction

6.1 Motivation

The categorization of biological objects is one of the fundamental and traditional tasks of the life sciences. For instance, the categorization of organisms into a hierarchical "Tree of life" leads to a complex model that summarizes not just the taxonomic relationships between the species, but also the putative time-course of evolution as we understand it today. With the advent of molecular biology in the 1970's, the categorization of genes and proteins itself became an important subject of research. Sequences of individual proteins can, for instance, be compared using string distance measures, and one can build trees that closely resemble the hypothetical "Tree of life". The categorization of protein structures, on the other hand, began from a different perspective: protein structures reveal a few fundamental molecular arrangements (like alpha-helices and beta-sheets) that can combine in a variety of ways and give rise to characteristic molecular shapes. Finally, the recent advent of genomics research - the wholesale analysis of the gene and protein repertoire of a species - led to yet another perspective with an emphasis on biological function. According to this approach, the known genes/proteins are categorized into *a priori* determined, empirical categories that reflect our current knowledge on the cellular and biochemical functions. As proteins carry many, perhaps most of the known biological functions, they play a prominent role in the functional analysis of genomes.

The methods of protein classification fall into three broad categories: i) Methods based on pairwise comparison, i.e. ones that work by comparing an unknown object (protein sequence or structure) with members of an a priori classified database of protein objects. The results are ranked according to the similarities and the strongest similarities are evaluated in terms of biological or statistical significance, after which a query is assigned to the class of the most similar object. ii) Methods based on consensus (or aggregate) descriptions, i.e. ones that are used to analyze distant sequence similarities that cannot readily be determined based on a simple similarity analysis. Here we first prepare a consensus description for all the classes of a protein sequence database, then we compare the unknown query with each of the consensus descriptions. As with

the previous methods, the strongest similarities are evaluated and used to assign the protein to a given class. There are various methods for preparing consensus descriptions, including regular expressions, frequency matrices and Hidden Markov Models. The above methods are described in textbooks and are periodically reviewed. iii) A more recent type of protein classification methods attempts to use an external source of knowledge in order to increase the classification sensitivity. The external source of knowledge is the phylogenetic classification of an organism, i.e. the knowledge that is accumulated in the fields of taxonomy and molecular phylogeny. This approach is called phylogenomics (for a recent review, see [76]) and is closely linked to the notions of orthologous (proteins that share both ancestry and function) and paralogues (proteins that share a common ancestry but carry different functions). The practical goal of phylogenomics is to reveal the true orthologous relationships and use them in the process of classification. Like ii), phylogenomic methods are used for distant similarities that cannot be decided by simple comparisons like those mentioned in i).

The aim of the present part of this thesis is to describe some protein classification algorithms that make use of tree structures. The chief difficulties of protein classification arise from the fact that the databases are large, noisy, heterogeneous and redundant; and that the classes themselves are very different in terms of most of their characteristics; that the assignments are often uncertain. For these reasons there is a constant need for better and faster algorithms that can cope with the growing datasets, and tree-based approaches are promising in this respect [77]. Even though trees are often used in molecular phylogenies and phylogenomics, the motivation of our work is quite different since we are not trying to reveal or to use the taxonomic relationships between species or proteins. We employ trees - especially weighted binary trees - as a simple and computationally inexpensive formalism to capture the hidden structure of the data and to use it for protein classification .

The rest of this chapter is concerned with describing of the tools we will use in our experiments. First, we will describe the sequence comparison methods and their parameter settings. Afterwards we will overview the protein classification task. Lastly, we will introduce the way we applied ROC analysis in model evaluation.

6.2 Description of classification task (Binary vs multi-class classification)

The protein classification tasks are mainly multi-class problems. This means that we have to categorize the objects or sequences into more than two classes. But we are mainly interested in the identification of one particular class, because it is convenient way of comparing the performance of models we investigate here. That is why we derive the multi-class approach to several different binary classification tasks. In a one-versus-all classification approach one particular class is treated as the positive class while the rest of the samples are treated as negatives (the binary classification task).

We should mention here that this derivation causes some problems with the model

evaluation process. Because in this one-versus-all classification the classes can be very imbalanced – i.e. we have only a few positive samples, while we have enormous number of negative samples. The traditional model evaluation metric like accuracy cannot measure the performance of models in an objective way, because if the model classified all elements as negative, then we would attain an acceptable accuracy. This problem typically comes up in text-mining [78]. In this field this problem is treated by applying an F-measure, which is the weighted harmonic mean of precision and recall. The traditional F-measure or balanced F-score is:

$$F = \frac{2(precision * recall)}{precision + recall} \tag{6.1}$$

In addition, the error rate is also a wide-spread model evaluation metric in many fields where machine learning has been applied. For example, the error rate - which is the fraction of errors (false positives and false negatives) within all the predictions. Thus *ER = 1-accuracy*
But in protein classification we consider the ranking performance of the different models, as well. Thus we will use ROC analysis like that describes in a later section in this chapter.

6.3 Sequence comparison algorithms

Version 2.2.4 of the BLAST program [1] had a cutoff score of 25. The Smith-Waterman algorithm [2] we used was implemented in MATLAB [66], while the program implementing the local alignment kernel algorithm [79] was obtained from the authors of the method. Moreover, the BLOSUM 62 matrix [28] was used in each case.

Compression based distance measures (CBMs) were used in the way defined in Section 3.4. We used in our experiments the LZW algorithm and the PPMZ algorithm. The LZW algorithm was implemented in MATLAB while the PPMZ2 algorithm was downloaded from Charles Bloom's homepage (http://www.cbloom.com/src/ppmz.html).

6.4 A brief applicability survey on tree building methods

Distance-based or the distance matrix methods of tree-building are fast and quite suitable for protein function prediction. The general idea behind each is to calculate a measure of the similarity between each pair of taxons, and then to find a tree that predicts the observed set of similarities as closely as possible. In our study we used two popular algorithms, the Unweighted Pair-Group Method using Arithmetic Averages (UPGMA) [10], and the Neighbour-Joining (NJ) algorithm [9]. Both algorithms here are based on hierarchical clustering. UPGMA employs an agglomerative algorithm which assumes that the evolutionary process can be represented by an ultrametric tree: or, in other words, that it satisfies the "molecular clock" assumption. On the other hand,

NJ is based on divisive clustering and produces additive trees. The time complexity of both methods is $O\left(n^3\right)$.

Biologists use a whole arsenal of sophisticated methods for building binary phylogenetic trees. One of the most popular ones is the Neighbor-Joining [9] and its more recent variants BioNJ [80] and Weighbor [81], which are known to produce consistent trees in $O(N^3)$ time provided we have additive distance. Since we needed to construct a tree for each query protein, we looked for fast and sensitive equivalents and chose the FastME algorithm that uses the Greedy Minimum Evolution tree construction method with Nearest Neighbor Interchange operator, and requires only $O(N^2)$ time [82]. The performance of FastME compares favorably with that of other, state-of-the-art algorithms [82]. We used the C implementation of FastME downloaded from http://www.ncbi.nlm.nih.gov/CBBresearch/Desper/FastME.html.

6.5 Performance evaluation methods

The fundamental use of ROC analysis is its application in binary (or two-class) classification problems. A binary classifier algorithm maps an object (for example an un-annotated object, in this sequence of 3D structure) into one of two classes, that we usually denote by $+$ and $-$, respectively. Generally, the parameters of such a classifier algorithm are derived from training on known + and - examples, then the classifier is tested on + and - examples that were not part of our training sets (Figure 9.2.1)

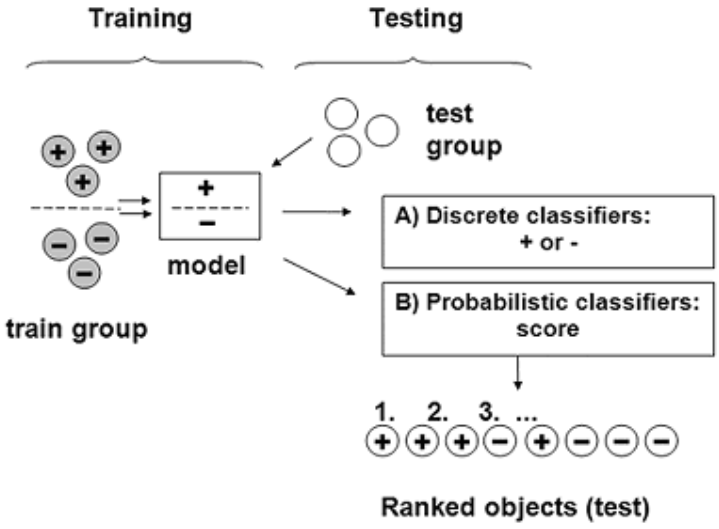
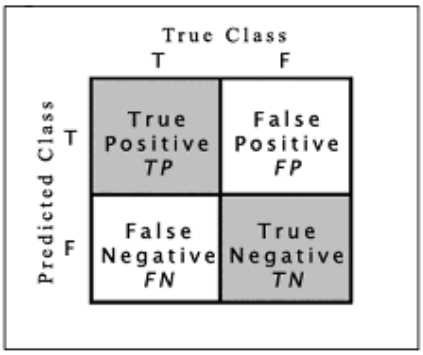


Figure 6.1: Binary classification. Binary classifiers algorithms (models, classifiers) that are capable of distinguishing two classes are denoted by + and -. The parameters of the model are determined from known + and - examples, this is the training phase. In the testing phase, test examples are given to the predictor. Discrete classifiers can assign only labels (+ or) to the test examples. And probabilistic classifiers assign a continuous score to the text examples which can be used for ranking.

A discrete classifier predicts only the class to which a test object belongs. There are four possible outcomes, that is true positive if the instance is + and it is correctly

classified as +, false negative if it is predicted as -, true negative if the instance is - and it is counted as -, and false positive if it is incorrectly predicted as -. If we evaluate a set of objects, we can count the outcomes and prepare a confusion matrix (also known as a contingency table), a two-by-two table that shows the classifier's correct decisions on the major diagonal and the errors off this diagonal (see Figure 9.3, left). Alternatively, we can construct various numeric measures that characterize the accuracy, the sensitivity and the specificity of the test (Fig. 9.3, right). These quantities lie between 0 and 1, and can be interpreted as probabilities. For instance, the false positive rate is the probability that a negative instance is incorrectly classified as being positive. Many similar indices have been reviewed in [83] and [84].

Probabilistic classifiers, on the other hand, return a score which is not necessarily a sensu stricto probability, but represents the degree to which an object is a member of a class rather than of the other one [85]. We can use this score for ranking a test set of objects, and a classifier works correctly if the positive examples are on top of the list. In addition, one can apply a decision threshold value to the score, say above which the prediction is considered positive. In such a way we changed the probabilistic classifier into a discrete classifier. Naturally, we can select different threshold values, and in this way for one probabilistic classifier we can generate an (infinitely long) series of discrete classifiers.



$$ACC = accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$SP = specificity = \frac{TN}{FP + TN}$$

$$TPR = sensitivity = \frac{TP}{TP + FN}$$

$$FPR = (1 - specificity) = \frac{FP}{FP + TN}$$

Figure 6.2: The confusion matrix and a few performance measures TP, TN, FP, FN are the number of true positives, true negatives, false positives and false negatives in a test set, respectively. TPR is the true positive rate or sensitivity, FPR is the false positive rate. A ROC curve is a TPR vs. FPR plot.

An ROC curve (Figure 9.3) is obtained by selecting a series of thresholds, and plotting sensitivity on the y axis versus 1-specificity on the x-axis (using the terms of Figure 2, it is a TPR vs. FPR plot). The output of our imaginary classifier is the ranked list shown on the left hand side of the figure. We can produce the ROC curve shown in bottom left of the figure by varying a decision threshold between the minimum and maximum of the output values, and plotting the FPR (1 - specificity) on the x-axis and the TPR (sensitivity) on the y-axis. (In practice, we can change the threshold so as to step on the next output value, in such a way we will create one point for each output value). The empirical ROC curve generated for this small test set is a step function, which will approach a continuous curve for large test sets.

Each point in this curve corresponds to a discrete classifier that can be obtained by using a given decision threshold. For example, when the threshold is set to 0.6, the True Positive Rate is 0.7 and the False Positive Rate is 0.1. An ROC curve is thus a two-dimensional graph that visually depicts the relative trade-offs between the errors (false positives) and benefits (true positives) [85]. We can also say that an ROC curve characterizes a probabilistic classifier, and each point of this curve corresponds to a discrete classifier. Interpretation of ROC curves A ROC curve can be interpreted either graphically or numerically, as schematically shown in Figure 9.3. A perfect probabilistic classifier corresponds to the top ROC curve indicated in a dashed line. Such a classifier assigns higher scores to the positives than to the negatives, so the positives will be on top of the ranked list (Table 9.3, b). This curve is rectangular and its integral, the "area under the ROC curve (AUC or AUROC) equals to 1. The dotted diagonal line corresponds to a "random classifier" that would give out random answers, irrespective of the input. The integral (AUC value) of this curve is 0.5 (Table 9.3, f). A correct classifier has a ROC curve above the diagonal and an $AUC > 0.5$. On the other hand, classifiers that consistently give the opposite predictions, ("anticorrelated" classifiers) give ROC curves below the diagonal, and AUC values between zero and 0.5 (Table 9.3, g, h), [1].

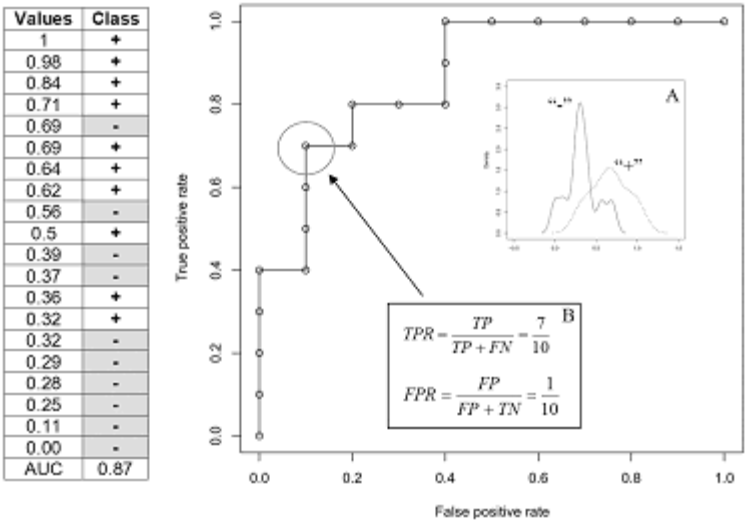


Figure 6.3: Constructing a ROC curve from ranked data. The TP,TN, FP, FN values are determined by comparing values to a moving threshold, an example of which is shown by an arrow in the ranked list (left). Above the threshold + data items are TP, - data items are FP. Therefore a threshold of 0.6 produces the point $FPR = 0.1$, $TPR = 0.7$ as shown in inset B. The plot is produced by moving the threshold through the entire range. The data were randomly generated based on the distributions shown in inset A.

From a mathematical point of view, the AUC can be viewed as the probability that a randomly chosen positive instance would be ranked higher than a randomly chosen negative instance that is equivalent to the two sample the Wilcoxon rank-sum statistic [2]. Alternatively, the AUC can be be interpreted either as the average sensitivity over

Table 6.1: Benchmark results of the cascade oscillators model

all false positive rates or as the average specificity over all sensitivities [3]. Note that AUC_n values (described below under pairwise comparison), cannot be interpreted in this fashion.

Ranks	a	b	c	d	e	f	g	h
1	+	+	-	-	-	-	-	-
2	+	+	+	-	-	-	-	-
3	+	+	+	+	-	-	-	-
4	+	+	+	+	-	-	-	-
5	-	-	+	+	-	-	-	-
6	+	-	-	+	+	-	-	-
7	+	-	-	-	+	-	-	-
8	+	-	-	-	+	-	-	-
9	-	-	-	-	+	+	-	-
10	+	-	-	-	-	+	-	-
11	-	-	-	-	-	+	-	-
12	-	-	-	-	-	+	+	-
13	+	-	-	-	-	-	+	-
14	+	-	-	-	-	-	+	-
15	-	-	-	-	-	-	+	-
16	-	-	-	-	-	-	-	-
17	-	-	-	-	-	-	-	+
18	-	-	-	-	-	-	-	+
19	-	-	-	-	-	-	-	+
20	-	-	-	-	-	-	-	+
AUC:	0.87	1.00	0.93	0.88	0.70	0.50	0.30	0.00

In practice, the AUC is often used as a single numerical measure of ranking performance. We should mention that ranking is dependent on the call distribution of the ranked set, so one cannot set an absolute threshold above which the ranking is good. In general, a high AUC value does not guarantee that the top ranking items will be true positive, as shown on synthetic data in Table I.

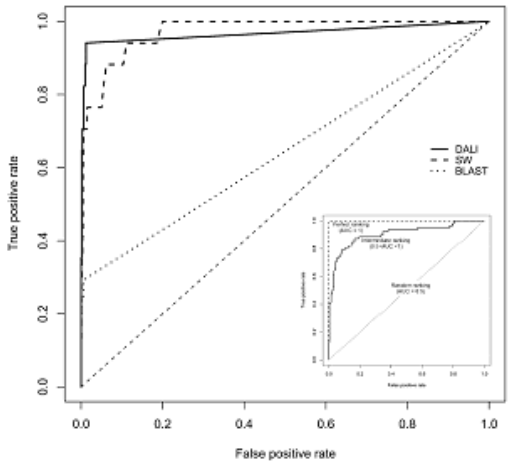


Figure 6.4: Examples of ROC curves calculated by pairwise sequence comparison using BLAST [1], Smith-Waterman [2] and a structural comparison using DALI [3]. The query was Cytochrome C6 from B. pasteurii, the + group were the other members of the Cytochrome C superfamily, the - set was the rest of the SCOP40mini dataset, taken from record PCB00019 of the Protein Classification Benchmark collection [4]. The diagonal corresponds to the random classifier. Curves running higher indicate a better classifier performance.

Chapter 7

Basic Tree-Based Models

7.1 Introduction

The algorithms described in this chapter belong to the broad area of protein classification, which have been summarized in several recent reviews and monographs [86]. In particular, we will employ the tools developed for protein sequence comparison that are now routinely used by researchers in various biological fields. The Smith-Waterman [2] and the Needleman-Wunsch algorithms [31] are exhaustive sequence comparison algorithms, while BLAST [1] is a fast heuristic algorithm. All of these programs calculate a similarity score that is high for similar or identical sequences and zero or below some threshold for very different sequences. Methods of molecular phylogeny build trees from the similarity scores obtained from the pairwise comparison of a set of protein sequences. The current methods of tree building are summarized in the splendid textbook by J. Felsenstein[21]. One class of tree-building methods, the class of so-called distance based methods, is particularly relevant to our work since we use one of the simplest method, namely Neighbour-Joining (NJ) [9], to generate trees from the data.

Protein classification supported by phylogenetic information is sometimes termed phylogenomics [18; 76]. The term covers an eclectic set of tools that combine phylogenetic trees and external data-sources in order to increase the sensitivity of protein classification [76]. Jonathan Eisen's review provides a conceptual framework for combining functional and phylogenetic information and describes a number of cases where functions cannot be predicted using sequence similarity alone. Most of the work summarized by Eisen is devoted to the human evaluation of small datasets by hand. The first automated annotation algorithm was introduced by Zmasek and Eddy [25], who used explicit phylogenetic inference in conjunction with real-life databases. Their method applies the gene tree and the species tree in a parallel fashion, and it can infer speciation and duplication events by comparing the two distinct trees. The worst case running time of this methods is $O(n^2)$, and the authors used the COG dataset [87] to show that their method is applicable for automated gene annotation.

Not long ago Lazareva-Ulitsky et. al. employed an explicit measure to describe the compatibility of a phylogenetic tree and a functional classification [88]. Given a

phylogenetic tree overlaid with labels of functional classes, the authors analyzed the subtrees that contain all members of a given class. A subtree is called perfect if its leaves all belong to the one functional class and an ideal phylogenetic tree is made up of just perfect subtrees. In the absence of such a perfect subdivision, one can establish an optimal division i.e. one can find subtrees that contain the least "false" labels. The authors defined a so-called tree measure that characterizes the fit between the phylogenetic tree and the functional classification, and then used it to develop a tree-building algorithm based on agglomerative clustering. For a comprehensive review on protein classification, see [76].

The rest of this chapter is structured as follows. Section 7.2 provides a brief overview of the datasets we used. Afterwards sections 7.3 and 7.4 respectively describe the two algorithms called *TreeNN* and *TreeInsert*. *TreeNN* is based on the concept of a distance that can be defined between leaves of a weighted binary tree. It is a pairwise comparison type algorithm (see i) above), where the distance function incorporates information encoded in a tree structure. Given a query protein and an *a priori* classified database, the algorithm first constructs a common tree that includes the members of the database and the query protein. In the subsequent step the algorithm attempts to assign labels to an unknown protein using the known class labels found in its neighborhood within the tree. A weighting scheme is applied, and the class label with the highest weight is assigned to the query. *TreeInsert* on the other hand is based on the concept of tree insertion cost, this being a numerical value characterizing the insertion of a new leaf at a given point of a weighted binary tree. The algorithm finds the point with minimum insertion cost in a tree. *TreeInsert* uses the tree as a consensus representation so it is related to the algorithms described above in ii). Given an unknown protein and protein classes represented by precalculated weighted binary trees, the query is assigned to the tree into which it can be inserted at the smallest cost. In the description of both algorithms we first give a conceptual outline that summarizes the theory as well as its relation to the existing approaches i-iii. This is followed by the formal description of the algorithm, the principle of the implementation, and some possible heuristic improvements. Then we round off this chapter with a brief discussion and some conclusions in Section 7.5.

7.2 Datasets

In order to characterize of the tree-based classifier algorithms described in this section we designed classification tasks. A classification task is a subdivision of a dataset into +train, +test, -train and -test groups. Here we used two datasets.

Dataset A was constructed from evolutionarily related sequences of an ubiquitous glycolytic enzyme, 3-phosphoglycerate kinase (3PGK, 358 to 505 residues in length). 131 3PGK sequences were selected which represent various species of the Archaean, Bacterial and Eukaryotic superkingdoms [89]. Ten classification tasks were then defined on this dataset in the following way. The positive examples were taken from a given superkingdom. One of the phyla (with at least five sequences) was the test set while the remaining phyla of the kingdom were used as the training set. The negative set

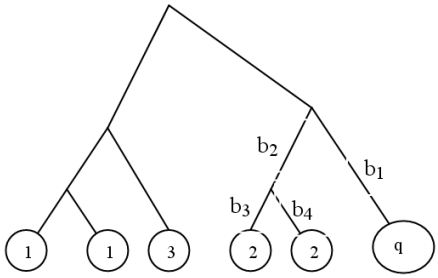


Figure 7.1: A weighted tree of proteins overlayed with class labels.

contained members of the other two superkingdoms and were subdivided in such a way that members of one phylum could be either test or train.

Dataset B is a subset of the COG database of functionally annotated orthologous sequence clusters [87]. In the COG database, each COG cluster contains functionally related orthologous sequences belonging to unicellular organisms, including Archaea, Bacteria and unicellular Eukaryota. Of the over 5665 COGs we selected 117 that contained at least 8 eukaryotic sequences and 16 additional prokaryotic sequences (a total of 17973 sequences). A separate classification task was defined for each of the 117 selected COG groups. The positive group contained the Archaean proteins randomly subdivided into +train and +test groups, while the rest of the COG was randomly subdivided into -train and -test groups. In a typical classification task the positive group consisted of 17 to 41 Archaean sequences while the negative group contained 12 to 247 members, both groups being subdivided into equal test and train groups.

7.3 *TreeNN*: Protein classification via neighborhood evaluation within weighted binary trees

7.3.1 Conceptual outline

Given a database of a priori classified proteins that are compared to each other in terms of a similarity/dissimilarity measure[1], and a query protein that is compared to the same database in terms of the same similarity/dissimilarity measure, one can build a phylogenetic tree that will contain proteins in each leaf. If we now assign the known class labels to the proteins, all leaves except the unknown query will be labelled, as schematically shown in Figure 7.1.

First let us denote the length of the unique path between two leaves L_i and L_j of a phylogenetic tree T by $p(L_i, L_j)$. Here $p(L_i, L_j)$ is an integer representing the number of edges along the path between L_i and L_j. We can define the closest neighbourhood $K(L)$ of a leaf L as the set of leaves for which there is no L_j leaf such that $p(L, L_j) < p(L, L_i)$. For instance the closest neighbours of q in Figure 7.1 are both members of

[1] "Similarity measures" and "dissimilarity measures" are inversely related: a monotone decreasing transformation of a similarity measure leads to a dissimilarity measure.

class 2 and are three steps apart from q. These leaves are parts of the 3-neighbourhood of q (i.e. the set of leaves for which the path between q and them at the most 3). If the tree is a weighted binary tree we can define the leaf distance $D^T(L_i, L_j)$ between two leaves L_i and L_j as the sum of the branch-weights along the unique path between L_i and L_j. For instance the leaf distance of q from one of its closest neighbors q in the tree is $b_1 + b_2 + b_3$. Finally let us suppose that we know the value of a pairwise similarity measure (such as a BLAST score) between any pair of leaves L_i and L_j, whose value will be denoted by $s(L_i, L_j)$, and that we build a weighted binary tree using the $s(L_i, L_j)$ values. Within this tree we can also calculate the value of the leaf distance $D^T(L_i, L_j)$.

The *TreeNN* algorithm is a weighted nearest neighbour method that applies as weights a similarity/dissimilarity measure between the proteins constituting the tree and is calculated within the closest neighbourhood of the query within the tree. More precisely, let us assume that we have leaves from m different classes and an indicator function $I : \{L_1, ..., L_n\} \rightarrow \{1, ..., m\}$ that assigns the class labels to the proteins represented by the leaves of the tree. The aggregate similarity measure $R(j, L_q)$ of each of the m classes ($j \in \{1, ..., m\}$) will be an aggregate of the similarity measures or leaf distances obtained between the query on one hand and the members of the class within the closest neighbourhood on the other, calculated via an aggregation operator Θ (such as the sum, product or maximum):

$$R(j, L_q) = \underset{L_i \in K(L) \wedge I(L_i) = j}{\Theta} s(L_i, L_q) \tag{7.1}$$

or

$$\widetilde{R}(j, L_q) = \underset{L_i \in K(L) \wedge I(L_i) = j}{\Theta} D^T(L_i, L_q) \tag{7.2}$$

The first aggregated value (Eq. (7.1)) for classes is calculated using the original similarity values. This implementation just utilized the topology of the phylogenetic tree, while the second implementation (Eq. (7.2)) also takes into account the edge lengths.

We calculate aggregate measures for each of the classes and the class with the highest weight will be assigned to the query L_q. This analysis is similar to that for the widely used kNN principle, the difference being that we restrict the analysis to the tree neighbourhood of the query (and not simply to the k most similar proteins) and we can use a leaf distance, as shown in Eq. (7.1). As for as the aggregation operator, we could for instance use summation, but we can also use the average operator or the maximum value operator.

In order to increase the influence of the tree structure on the ranking, we introduce a further variant of *TreeNN*, in which the similarity measures $s(L_i, L_j)$ are divided by the path lengths between L_i and L_j. In this manner the weighted aggregate similarity measure becomes

$$W(j, L_q) = \underset{L_i \in K(L) \wedge I(L_i)=j}{\Theta} \left(\frac{s(L_i, L_q)}{p(L_i, L_q)} \right) \tag{7.3}$$

or

$$\widetilde{W}(j, L_q) = \underset{L_i \in K(L) \wedge I(L_i)=j}{\Theta} \left(\frac{D^T(L_i, L_q)}{p(L_i, L_q)} \right) \tag{7.4}$$

This formula ensures that the leaves further away from L_q within the tree will have a smaller influence on the classification than the nearer neighbours.

7.3.2 Description of the algorithm

Input:

- A distance matrix containing the all-vs.-all comparison of a dataset, consisting of a query protein and an a priori classified set of data.

Output:

- A class label assigned to the query protein

First, a weighted binary tree is built from the data. The leaves of this tree are proteins and we select the set of closest tree-neighbours (minimum number of edges from the query). Then we apply a classification rule that might be one of the following:

TreeNN:	Assigns to the query the class label with the highest aggregate similarity calculated according to Eq. (7.1) or (7.2).
Weighted TreeNN:	Assigns to the query the class label with the highest aggregate similarity calculated according to Eq. (7.3) or (7.4).

The time complexity of the algorithm mainly depends on the tree-building part. For example, the Neighbour-Joining method has an $O(n^3)$ time complexity. We have to construct a tree with $n+1$ leaves as each protein will be classified, hence this algorithm has an $O(tn^3)$ time complexity overall where t denotes the cardinality of the test set. Finding the closest tree-neighbours for a leaf can be carried out in linear time, hence it does not cause any extra computational burden.

Its use in classification. The algorithm can be directly used both in two-class and multi-class classification. The size of the database influences both the time requirement of the similarity search and that of the tree-building process. The latter is especially important since the time-complexity of tree building is $O(n^3)$. We can substantially

speed up the computation if we include into the tree just the first r similarity/dissimilarity neighbours of the query (e.g. the first 50 BLAST neighbours). On the other hand class imbalance can cause an additional problem, since an irrelevant class that has many members can easily outweigh smaller classes. An apparently efficient balance heuristic is to build the tree from the members of the first t $(t \leq 10)$ classes nearest to the query, where each class is represented by a maximum of r $(r \leq 4)$ members.

7.3.3 Implementation

A computer program was implemented in MATLAB that uses the NJ algorithm as encoded in the Bioinformatics Toolbox package [66]. The detailed calculation has four distinct steps:

1. An-all-vs. all distance matrix is calculated from the members of an *a priori* classified database using a given similarity/dissimilarity score (like BLAST or Smith-Waterman) and the results are stored in CVS (Comma Separated Values).

2. The query protein is compared with the same database and the same similarity/dissimilarity score, and the first r sequences are selected for tree-building, choosing one of the heuristics mentioned above.

3. A small $([r+1] \times [r+1])$ distance matrix is built using the precomputed data of the database on the one hand and the query vs. database comparison on the other, and a NJ tree is built.

4. The query's label is assigned using the *TreeNN* algorithm in the way described in Section 7.3.2.

This implementation guarantees that the all-vs.-all comparison of the database is carried out just once.

7.3.4 Performance evaluation

The performance of *TreeNN* was evaluated by ROC analysis and error rate calculations in the way described in the previous chapter (Section 6.5). For comparison we also include the results obtained by simple nearest neighbour analysis (1NN). In each table below the best scores of each set are given in bold, and the columns with the heading *Full* concern the performances of *TreeNN* without a heuristic (i.e. we considered all the elements of the training set). Here we apply the *TreeNN* methods for a two class problem, thus the parameter t is always equal to 2. For aggregation operators we tried out the sum, the average and the maximum operators (but the results are not shown here), and the latter (more precisely, maximum for similarity measures and minimum for distance measures) had a slightly but consistently superior performance, so we used this operator to generate the data shown below. For the calculation of the class weights we applied the scoring scheme that is based on the similarity measures given in Eq.

Table 7.1: ROC values of *TreeNN* with and without a heuristic on the COG and 3PGK datasets.

	1NN	*TreeNN*		
		$Full$	$r = 3$	$r = 10$
COG				
BLAST	0.8251	**0.8381**	0.8200	0.7226
Smith-Waterman	0.8285	0.8369	**0.8438**	0.7820
LAK	0.8249	0.8316	**0.8498**	0.7813
LZW	**0.8155**	0.7807	0.7750	0.7498
PPMZ	0.7757	0.8162	**0.8162**	0.7709
3PGK				
BLAST	0.8978	0.9580	**0.9699**	0.9574
Smith-Waterman	0.8974	0.9582	**0.9716**	0.9587
LAK	0.8951	0.9418	**0.9688**	0.9641
LZW	0.8195	0.8186	**0.9040**	0.8875
PPMZ	0.8551	0.9481	**0.9556**	0.7244

(7.1) for *TreeNN* and Eq. (7.3) for *Weighted TreeNN*. Tables 7.1 and Table 7.2 show the *TreeNN* results for ROC analysis and error rate calculations, respectively. In the next two tables (Table 7.3 and 7.4) we show the performance of the *Weighted TreeNN* using the same settings as that used for *TreeNN* in Tables 7.1 and 7.2. The results, along with the time requirements (wall clock times) are summarized in Tables 7.5. The time requirements of *Weighted TreeNN* is quite similar to that of *TreeNN* because the two differ only in the calculation of class aggregate values (cf. Eqs. (7.3) and (7.4)).

Table 7.2: ROC values of Weighted *TreeNN* with and without a heuristic on the COG and 3PGK datasets.

	1NN	*TreeNN*		
		$Full$	$r = 3$	$r = 10$
COG				
BLAST	0.8251	**0.8454**	0.8206	0.8016
Smith-Waterman	0.8285	0.8474	**0.8492**	0.8098
LAK	0.8249	0.8417	**0.8540**	0.8133
LZW	0.8195	**0.9356**	0.9040	0.9228
PPMZ	0.8551	**0.9797**	0.9673	0.8367
3PGK				
BLAST	0.8978	**0.9760**	0.9589	0.9579
Smith-Waterman	0.8974	**0.9761**	0.9547	0.9510
LAK	0.8951	0.9612	**0.9719**	0.9354
LZW	**0.8195**	0.7365	0.7412	0.8183
PPMZ	**0.8551**	0.8140	0.8018	0.7767

The above results reveal a few general trends. First, *TreeNN* and its weighted version outperforms the 1NN classification in terms of the error rate. We should mention here that this improvement in error rate is apparent even when we use a heuristic. As

Table 7.3: Error rates of *TreeNN* with and without a heuristic on the COG and 3PGK datasets.

	1NN	*TreeNN*		
		$Full$	$r = 3$	$r = 10$
COG				
BLAST	14.7516	**10.3746**	11.6270	15.1469
Smith-Waterman	13.4940	10.5381	**9.9996**	13.0470
LAK	13.3817	10.8644	**9.7976**	12.3784
LZW	16.7301	**13.2106**	13.9285	14.4467
PPMZ	15.0174	**11.6331**	11.9598	13.2246
3PGK				
BLAST	42.1046	35.4026	**32.2360**	35.8291
Smith-Waterman	42.1046	35.6582	**32.2360**	35.5694
LAK	42.0856	33.4081	**32.1928**	34.0542
LZW	36.5293	35.1731	33.8335	**30.4403**
PPMZ	34.6671	37.2146	**32.1706**	37.4445

for AUC, the results on the COG database are comparable with those of 1NN, both with and without a heuristic. Moreover they are noticeably better than 1NN on the 3PGK dataset. The fact that the precision improves while the time requirements are comparable with that of the very fast 1NN algorithm is a good sign and confirmation that our approach is a promising one (Table 7.5).

We calculated the time requirements for the methods we employed in a real life scenario. We first assumed that we had an *a prior* classified dataset containing 10000 elements. Applying the 1NN we simply needed to find the most similar protein in this dataset to the query protein, and the class label of this was assigned to the query protein. This approach did not need additional preprocessing step, so these process time requirements were just equal to the calculation of the similarity measure between the query and the *a prior* classified dataset. The *TreeNN* method however required some additional running time. This was because after the *TreeNN* had chosen the r most similar element from the known dataset (according to the heuristic in Section 7.3.2) it built up a phylogenetic tree for these elements. This step suggested some additional preprocessing time requirements. In our experiments the parameter was set to 100. As the experiments showed, applying *TreeNN* did not bring about any significant growth in time requirements.

Table 7.6 lists a comparison of the performance of the *TreeNN* algorithm when we used the original similarities/distances of proteins and when we used the leaf distances just according to Eqn. (7.2). The results of these tests clearly show that the performance of the classifiers was only marginally influenced by the measure (sequence similarity measure vs. leaf distances) we chose in the implementation of the algorithm.

Table 7.4: Error rates of the Weighted *TreeNN* with and without a heuristic on the COG and 3PGK datasets.

	1NN	*TreeNN*		
		Full	$r = 3$	$r = 10$
COG				
BLAST	14.7516	**10.3746**	11.6270	15.1469
Smith-Waterman	13.4940	10.5381	**9.9996**	13.0470
LAK	13.3817	10.8644	**9.7976**	12.3784
LZW	16.7301	**13.2106**	13.9285	14.4467
PPMZ	15.0174	**11.6331**	11.9598	13.2246
3PGK				
BLAST	42.1046	35.4026	**32.2360**	35.8291
Smith-Waterman	42.1046	35.6582	**32.2360**	35.5694
LAK	42.0856	33.4081	**32.1928**	34.0542
LZW	36.5293	35.1731	33.8335	**30.4403**
PPMZ	34.6671	37.2146	**32.1706**	37.4445

Table 7.5: Time requirements for the *TreeNN* method on the COG dataset in seconds.

Elapsed time(in second)/Method		1NN	*TreeNN(r=100)*
Preprocessing	BLAST	-	-
	Other	-	15.531
Evaluation	BLAST	0.223	0.223
Evaluation	Other	-	0.109

7.4 *TreeInsert*: Protein classification via insertion into weighted binary trees

7.4.1 Conceptual outline

Given a database of a priori classified proteins, we can build separate phylogenetic trees from the members of each of the classes. A new query protein is then assigned to the class to which it is nearest in terms of *insertion cost (IC)*. A query protein will then be assigned to the class whose *IC* is the smallest. First we note that insertion of a new

Table 7.6: The performance of the *TreeNN* using leaf distances and the original similarity measures.

Comparison of the *TreeNN*	*TreeNN*			
	Using similarities		Using leaf distances	
	ROC	Error Rate	ROC	Error Rate
BLAST	0.9699	35.4026	0,9509	36,978
Smith-Waterman	0.9582	35.6582	0,9529	36,979

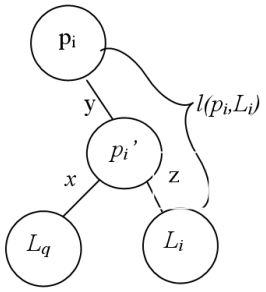


Figure 7.2: The insertion of the new leaf next to L_i.

leaf into a weighted binary tree is the "amount of fitting" into the original tree. In this algorithm we consider an insertion optimal if the query protein is the best suited compared to every other possible insertion. Second, note that *IC* can be defined in various ways using the terminology introduced in Section 7.3. The insertion of a new leaf L_q next to leaf L_i is depicted in Figure 7.2.
In this example we insert the new element L_q next to the i-th leaf of the phylogenetic tree T so we need to divide the edge between L_i and its parent into two parts with a novel inner point p_i'. According to Figure 7.2, we can express the relationship of the new leaf L_q to the other leaves of the tree in the following way: $D^T(L_j, L_q) = D^T(p_i, L_j) + y + z$ if $i \neq j$. The $D^T(L_i, L_q)$ leaf distance between the ith leaf and L_q is just equal to $x + z$. This extension step of the leaf distances means that all relations in the tree remain the same, and we have only to determine the new edge lengths x, y and z. The place of p_i' on the divided edge and the weights of the edge that are between L_q and its parent (denoted by x in Figure 7.2) have to be determined so that the similarities and the tree-based distances will be as close as possible. With this line of thinking we can formulate the insertion task as the solution of the following system of equations:

$$\min_{0 \leq x,y} \left(\sum_{j=1}^{n} \left(s(L_j, L_q) - D^T(L_i, L_q) \right) \right)^2 \tag{7.5}$$
$$\text{s.t.} \quad x + y = D^T(p_i, L_i)$$

This optimization task determines the value of the three unknown edge lengths x, y and z, and the constraints ensure that the leaf-distance between L_i and its parent remains unchanged. With this in mind, we can define the insertion cost for a fixed leaf.

Definition 7.1 *Let T be a phylogenetic tree and let its leaves be $L_1, L_2, ..., L_n$. The leaf insertion cost $IC(L_q, L_i)$ of a new leaf L_q next to L_i is defined as the branch length of x found by solving the optimisation task in Eq. (7.5).*

Our goal here is to find the position of the new leaf in T with the lowest leaf insertion cost. This is why we define the insertion cost of a new leaf for the whole tree using the Definition 7.1 in the following way:

Definition 7.2 *Let T be a phylogenetic tree and let its leaves be $L_1, L_2, ..., L_n$. The insertion cost $IC(L_q)$ of a new leaf L_q into T is the minimal leaf insertion cost for T:*

$$IC(L_q) = \min\{IC(L_q, L_1), ..., IC(L_q, L_n)\} \tag{7.6}$$

In preliminary experiments we tried several possible alternative definitions for the insertion cost *IC* (data not shown), then finally we chose the branch length x (Figure 7.2) as the definition. This value provides a measure of dissimilarity: it is zero when the insertion point is vicinal to a leaf that is identical with the query. The *IC* for a given tree is the smallest value of x found within the tree.

7.4.2 Description of the algorithm

Input:

- A weighted binary tree built using the similarity/dissimilarity values (such as a set of BLAST scores) taken between the elements of a protein class.
- A set of comparison values taken between a query protein on the one hand and the members of the protein class on the other, using the same similarity/dissimilarity values as we used to construct the tree. So for instance, when the tree was built using BLAST scores, the set of comparison values were a set of BLAST comparison values.

Output:

- The value of the insertion cost calculated according to Definition 7.2.

The algorithm will evaluate all insertions that are possible in the tree. An insertion of a new leaf next to an old one requires that the solution of an equation system that consists of n equations, where n is the number of leaves. This will have a time complexity of $O(n)$. The number of possible insertions for a tree having n leaves (i.e. we insert them next to each leaf) is n. Thus calculating the insertion for a new element has a time complexity of $O(n^2)$. One can reduce the time complexity using a simple empirical consideration: we just assume that the optimum insertion will occur in the vicinity of the r leaves that are most similar to the query in terms of the similarity/dissimilarity measure used for the evaluation. If we use BLAST, we can limit

the insertions to the r nearest BLAST neighbours of the query. This will reduce the time complexity of the search to $O(rn)$.

Its use in classification. If we have a two-class classification problem, we will have to build a tree both for the positive class and the negative class, and we can classify the query to the class whose *IC* is smaller. In practical applications we often have to classify a query into one of several thousand protein classes, such as the classes of protein domains or functions. In this case the class with the smallest *IC* can be chosen. This is a simple nearest neighbour classification which can be further refined by adding an *IC* threshold above which the similarities shall not be considered. In order to decrease the time complexity, we can also exclude from the evaluation those classes whose members did not occur among the r proteins most similar to the query. Protein databases are known to consist of classes very different in size. As the tree size does not influence the insertion cost, class imbalance will not represent a problem to *TreeInsert* when calculations are performed.

7.4.3 Implementation

We used the Neighbor-Joining algorithm for tree-building as given in the MATLAB Bioinformatics Toolbox [66]. In conjunction with the sequence comparison methods listed in Section 6.3, the programs were implemented in MATLAB. The execution of the method consists of two distinct steps, namely:

1. The preprocessing of the database into weighted binary trees and storage of the data in Newick file format [61]. For this step, the members of each class were compared with each other in an all-vs.-all fashion, and the trees were built using the NJ algorithm. For a large database like COG (51 groups 5332 sequences) the entire procedure takes 5.95 Seconds on a Pentium IV Computer (3.0 GHz processor).

2. First, the query is compared with the database using a selected similarity/dissimilarity measure and the data are stored in CSV file format. Next, the query is inserted into a set of class-representation trees, and the class with the optimal (smallest) *IC* value is chosen.

7.4.4 Performance evaluation

The performance of *TreeInsert* was evaluated via ROC analysis and via the error rate, as described in Section 6.5. For comparison we also include here the results obtained by simple nearest neighbour analysis (1NN). The results, along with the time requirements (wall clock times) are summarized in Table 7.9. Our classification tasks were the same as those in Section 7.3.4, thus the parameter t (number of given class) was always equal to 2. The dependence of the performance on the other tuneable parameter r (the number of elements per class) is shown in Tables 7.7 and 7.8.

Table 7.7: ROC analysis results (AUC values) for the *TreeInsert* algorithm on the COG and 3PGK datasets. Here several different implementations were used.

	1NN	*TreeNN*		
		$Full$	$r = 3$	$r = 10$
COG				
BLAST	0.8251	**0.8741**	0.8441	0.8708
Smith-Waterman	0.8285	**0.8732**	0.8474	0.8640
LAK	0.8249	0.8154	0.8276	**0.8734**
LZW	0.8155	0.7639	0.8243	**0.8316**
PPMZ	0.7757	0.8171	0.8535	**0.8682**
3PGK				
BLAST	0.8978	**0.9473**	0.8984	0.9090
Smith-Waterman	0.8974	**0.9472**	0.8977	0.9046
LAK	0.8951	**0.9414**	0.8851	0.9068
LZW	0.8195	**0.8801**	0.8009	0.8421
PPMZ	0.8551	0.8948	0.8646	**0.9123**

In most of the test cases *TreeInsert* visibly outperforms 1NN in terms of ROC AUC and error rate. What is more, the *TreeInsert* method achieves the best results when we consider all the possible insertions, not just those of the adjacent leaves. This probably means that the insertion cost is not necessarily correlated with the similarity measure between the proteins.

When we examined the classification process using TreeInsert we found that we needed to carry out a preprocessing step before we evaluated the method. This preprocessing step consisted of the building of the phylogenetic trees for each class in the training dataset. Following the testing scheme we applied in Section 7.3.4, we assumed that the training dataset contained 1000 classes, and the classes contained 100 elements on average. Thus this step caused a significant growth in the running time. But when we investigated this method from a time perspective we found that this extra computation cost belonged to the offline time requirements. The evaluation of this method hardly depended on the number of classes in question because we had to insert an unknown protein into the phylogenetic trees of the known protein family. Table 7.9 describes this dependency, where n here denotes the number of classes.

7.5 Discussion and conclusions

The problem of protein sequence classification is one of the crucial tasks in the interpretation of genomic data. Simple nearest neighbour (kNN) classification based on fast sequence comparison algorithms such as BLAST is efficient in the majority of the cases, i.e. up to $70 - 80\%$ of the sequences in a newly sequenced genome can be classified in a reassuring way, based on their high sequence similarity. A whole arsenal of sophisticated methods has been developed in order to evaluate the remaining $20 - 30\%$ of sequences that are often known as "distant similarities". The most popular current

Table 7.8: Error rate values for the *TreeInsert* algorithm on the COG and 3PGK datasets. As before, several different implementations were used.

	1NN	*TreeNN*		
		$Full$	$r = 3$	$r = 10$
COG				
BLAST	14.7516	**10.6127**	17.3419	17.3419
Smith-Waterman	**13.4940**	13.8218	17.9189	17.9189
LAK	13.3817	**11.3340**	15.9436	15.9436
LZW	16.7301	**13.8962**	20.0073	20.0073
PPMZ	15.0174	11.3386	**8.3167**	8.3167
3PGK				
BLAST	42.1046	**20.2009**	25.4544	35.7754
Smith-Waterman	42.1046	**20.3730**	24.7976	36.0115
LAK	42.0856	**20.2009**	25.8242	39.5036
LZW	36.5293	**15.7901**	37.0648	26.4240
PPMZ	34.6671	**14.4753**	32.3829	28.9935

Table 7.9: Time requirements of the *TreeInsert* methods on the COG dataset. Here n means the number of classes in question.

Elapsed time(in second)/Method		1NN	*TreeInsert(r=100)*
Preprocessing	BLAST	-	2232.54
	Other	-	1100
Evaluation	BLAST	0.223	0.223
Evaluation	Other	-	$0.029 * n$

methods are "consensus" descriptions (see ii) in the Introduction) that are based about the multiple alignment of the known sequence classes. A multiple alignment can be transformed either into a Hidden Markov model or a sequence profile; both are detailed, structured descriptions that contain sequence position-specific information on the multiple alignments. A new sequence is then compared with a library of such descriptions. These methods use some preprocessing that requires some CPU time as well as human intervention. Also the time of the analysis (evaluation of queries) can be quite substantial, especially when these are compared to BLAST runs. The golden mean of sequence comparison is to develop classification methods that are as fast as BLAST, but are able to handle the distant similarities as well.

The rationale behind applying tree-based algorithms is to provide a structured description that is simple and computationally inexpensive, but still may allow one to exceed the performance of simple kNN searches. *TreeNN* is a kNN type method that first builds a (small) tree from the results of the similarity search and then performs the classification in the context of this tree. *TreeInsert* is a consensus type method that requires a preprocessing time as well as an evaluation time. Both *TreeNN* and *TreeInsert* exceed the performance of simple similarity searches and this is quite promising for future practical applications. We should remark here however that the above comparisons were

made on very difficult datasets. On the other hand we used two-class scenarios, whereas the tasks in genome annotation are multiclass problems. Nevertheless, both *TreeNN* and *TreeInsert* can be applied in multiclass scenarios without extensive modifications so we are confident that they will be useful in these contexts. According to preliminary results obtained on the Protein Classification Benchmark collection [4] it also appears that, in addition to sequence comparison data, both algorithms can be efficiently used to analyse protein structure classification problems, which suggests that they might be useful in other fields as well, where items need to be classified.

Chapter 8

Propagational methods

8.1 Introduction

Propagation algorithms have gained importance in several areas of pattern recognition and information retrieval. Belief propagation or message passing [90; 91], the *PageRank* algorithm [92] and the power method [93] are all based on propagating information through a network that consists of nodes and edges. Several propagation algorithms were successfully used in practice, the PageRank algorithm used by the Google WEB surfer perhaps being the best known example.

Protein classification is a crucial task in genome annotation, and propagation algorithms were also successfully employed in this field [94–96]. The apparent advantage of this approach originates from the fact that classification does not just rely on a simple similarity measure but also on the entire network protein similarities. In fact, the first application of a *PageRank* style algorithm to protein classification [94] demonstrated that propagation yield a definite increase in classification efficiency. On the other hand, protein similarity networks are large, and they typically contain several hundred thousand proteins as nodes and several million pairwise similarity values as links, which makes propagation rather time-consuming. To overcome this, one possibility is to use smaller networks for the propagation, like bipartite graphs [95] or threshold graphs [97]. In this chapter we explore another avenue that instead of propagating on the entire network focuses on replacing a relevant part of the protein similarity network by a structured description. This approach is based on the well known observation that protein similarity networks are modular, i.e. they contain densely connected subgraphs around each protein class, wherein the biologically important similarities between members of the same class can be easily distinguished from the accidental similarities that exist between members of different classes [98; 99]. In addition, structured descriptions such as phylogenetic trees are known to capture the important features of a classification scheme, so they may serve as a natural noise reduction filter for propagation algorithms. Here we propose to replace (a selected part of) the protein similarity network with a binary tree on which the propagation will be carried out. Binary trees are sparse structures as compared to a full network of similarities, so there

is potential gain in speed. On the other hand, one has to build a binary tree, which is an extra burden, but as we will see, there is a substantial overall gain in computing time.

We will now describe two heuristic protein classification algorithms that use propagation on binary trees. *TreeProp-N* is modelled on the *PageRank/RankProp* philosophy. It uses an initial ranking vector based on the distance of the database entries from the query measured along the edges of the tree. This ranking is then updated by propagating information along the edges of the tree. *TreeProp-E* uses a different propagation principle: the initial data is stored in the weighted edges of the binary tree, and the weights afterwards are propagated to the neighboring edges by a simple update rule. We will use the fast tree-building algorithm FastME [82] to construct the initial tree for both algorithms.

The rest of this chapter is organized as follows. In the next section we will introduce the *PageRank* method and its applications to protein classification tasks (Section 8.2). Then we will introduce two new algorithms called *TreeProp-N* and *TreeProp-E* that use phylogenetic trees for the propagation (Section 8.3 and Section 8.4). The efficiency of our methods will then be compared on various real-life benchmark datasets, including protein sequences as well as protein 3D structures. Section 8.6 contains the a summary of the results and a discussion. All the other algorithms and datasets used here are described in the Chapter 6.

8.2 *PageRank* and its application to protein classification

Originally, the *PageRank* algorithm [92] was developed for information retrieval purposes. There are many other areas where this simple idea was adopted with success, including Natural Language Processing [100], Word Sense Disambiguation [101] and Protein Classification [94]. The common representation underlying these diverse fields is an a priori known similarity network of objects (i.e. a weighted graph where the points correspond to the objects, and the edges represent the similarities among them). Each similarity network can be represented as a stochastic matrix S in which each row sums up to one and which has no negative entry. An entry of the matrix S represents the similarities between two objects. *PageRank* converges to the stationary point of transformation S (i.e., the stationary distribution of those Markov chains where the transition matrix is S). We can express this ranking in an iterative fashion using the following update rule:

$$y(t+1) = Sy(t), \tag{8.1}$$

where $y(t+1)$ denotes the similarity score after t iterations. Since this rule corresponds to the ordinary power method [102], the convergence is fulfilled [93]. Moreover the $Y = \{y(0), y(1), \ldots, y(T), \ldots\}$ sequence will converge to the biggest eigenvector of the stochastic matrix S, which is - due to the Perron-Frobenius theorem - equal to 1.

We should note here that this method computes a single rank value for each object of the given network, i.e. this is not yet a query-based algorithm. Besides this the convergence of this process is known to depend on the gap between the first and second eigenvalues [102]. It may thus be worthwhile to explore the eigenvalue difference in order to estimate the necessary number of iterations before we use this approach on a similarity network. With biologists it is normal practice to have an unknown protein sequence which is compared to a database using a sequence comparison tool, like the Smith-Waterman (SW) algorithm [2] or BLAST algorithm [1]. The result is a ranking of the database entries according to a similarity score with respect to the query, and the top hits are evaluated either by an expert or by automated procedures. The query-based or personalized version of PageRank is more applicable for this type of a problem. Let us denote the protein entries of the database by $p_1, p_2, \ldots, p_N$ and the query protein by q. Now let's define a pairwise similarity measure between the ith and jth protein as $s(p_i, p_j)$ (typically this will be a BLAST or Smith Waterman similarity score). We can then arrange the $s(p_i, p_j)$ similarities into a matrix S, where the rows will be normalized so that their sum equals 1. After, with the help of matrix S, we can formulate the update rule of the so-called *personalized PageRank* method like so:

$$y(t+1) = y(0) + \alpha S y(t), \tag{8.2}$$

Here $y(0)$ is the initial vector of similarity scores and the α parameter is a constant in the range $[0, 1]$. This process converges to the y^* fix-point of Eq. 8.2, and it can also be calculated analytically by solving the $(I - \alpha S)^{-1} y(0) = y^*$ system of linear equations. But the size of matrix S is $N \times N$, so the analytical solution requires $O(N^3)$ time. On the other hand, we can adequately approximate y^* by $y(T)$ (i.e. the Tth element of the iteration sequence). The *personalized PageRank* algorithm was introduced for protein classification as the *RankProp* algorithm in the seminal paper of Weston et al [94].

A slightly different update rule was also introduced by Zhou et al in [97]:

$$y(t+1) = (1-\alpha) y(0) + \alpha S y(t), \tag{8.3}$$

It is easy to show that the y^* limit point of Eq. 8.3 is equal to $(1-\alpha)(I - \alpha S)^{-1} y(0)$. This form of PageRank leads to the same ranking as Eq. 8.2.

As restricting the size of the similarity network is a plausible way of speeding up the calculation, bipartite graphs [95] or threshold graphs [97] were both used for this. In the following sections we will present two novel algorithms that approach the problem of size restriction via the use of binary (phylogenetic) trees instead of a similarity network.

8.3 *TreeProp-N*: Propagation on the nodes of a binary tree

The main idea behind the *personalized PageRank* method is the application of a diffusion operation on a network of pairwise protein similarity, according to Eqs. 8.2-8.3. In this section we will introduce an alternative algorithm, namely *TreeProp-N*, where a similar diffusion operation is applied on an unrooted weighted phylogenetic tree that was built up from a database and the query protein q. We should mention here that the phylogenetic tree is a binary tree graph wherein the leaves of the tree correspond to the biological objects. In our work, we used the FastME algorithm [82] to construct the tree from the pairwise similarity scores computed between the elements. If we use similarity scores $s_{ij} \in [0, 1]$, then the tree will be built from a distance score expressed as $1 - s_{ij}$.

The propagation on the resulting tree in accordance with the update rule of the PageRank. The initial scores for each node (leaves and inner nodes) in the tree will be the shortest path between a given node and query element q. Since for the propagation we need similarity scores, rather than distance values we transform the shortest path values to similarity score $[0, 1]$ by dividing them with the maximal shortest path and subtracting the result from 1.

The propagation is carried out on the nodes of $T = (V, E)$, where $|V| = N+1$. An unrooted binary tree T with $N+1$ leaves has $2N$ nodes, so we can collect the shortest path from the q query point to the points of the T tree into a y vector of length $N+1$. According to the binary branching pattern of the phylogenetic tree, a leaf will only have one neighbor while an inner node will have three neighbors. Next we will denote the neighbors of a point p by $N(p)$. The update rule can be written as:

$$y_i(t+1) = (1-\alpha)y_i(0) + \alpha \sum_{p \in N(p_i)} w(p, p_i) y_p(t), \tag{8.4}$$

where $w(p, p_i)$ stands for the weight of the edge between p and p_i within the tree, and $y_i(t)$ means the propagated value of the point p_i after the t iterations. The α parameter lies between $[0, 1]$, just like in Eq. 8.3. This parameter sets the balance between the effect of tree-structure on the one hand, and the effect ranking on the other: higher values will emphasize the former over the latter.

The convergence of the propagation is ensured if the outgoing edge weights from a point sum up to one, which is provided by a normalization step. We should add that the resulting matrix has at most three elements in each of its rows, so the calculation can be carried out in $O(tn)$ time, where t is the number of iterations.

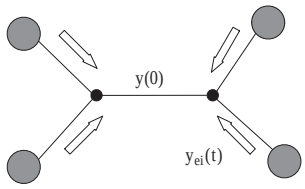


Figure 8.1: The process of the local operations on the edges. Each neighboring edges of an internal edge sends "messages" to its neighbor, and the magnitude of the message is proportional to the propagated edge weight.

8.4 *TreeProp-E*: Propagation on the edges of a binary tree

In a conventional propagation algorithm, the nodes of a similarity networks send messages to each other, and the magnitude of the messages is then proportional to the similarity between the sender and receiver. Here we introduce a novel approach where the edge weights of a binary tree will be propagated. The idea behind this approach is similar to the original *PageRank* concept, because the weight of an edge will be determined by the weight of its neighbors.

In the first step we build up an T unrooted weighted phylogenetic tree using query protein q and the known entries of a database. This tree has $N+1$ leaves and $2N-1$ edges. Since we propagate on the edges the $y(i)$ vectors have a length of $2N-1$. In a tree we will call two edges adjacent if they have a common endpoint. In an unrooted binary tree an edge pointing to a leaf has two adjacent edges, while an interior edge has four adjacent edges. Let us denote the set of adjacent edges with edge e by $N(e)$.

Now let $y(0)$ be the initial value of the edge lengths of the tree. Then we can apply the following simple propagation rule:

$$y_i(t+1) = (1-\alpha)y_i(0) + \frac{\alpha}{|N(e_i)|}\sum_{e_j \in N(e_i)} y_{e_i}(t), \tag{8.5}$$

which means that the score of an edge in step $(t+1)$ will depend on the mean value of the adjacent edge weights at step t as well as on its original weight at $t=0$. This update will be repeated a predetermined number of times, after which the ranking with respect to the query q can be calculated using the weighted path lengths between the query and the other proteins in the tree, calculated from the updated weights.

The convergence proof of PageRank can be directly applied to TreeProp-E. If we rewrite the propagation rule in Eq. 8.5 into the matrix form, as in the case of PageRank (Eq. 8.3), we get a non-negative real matrix and the Frobenius-theorem can be applied to it. And since its row sum is less than one, its spectral radius is also less than one so the convergence is again ensured.

8.5 Experiments

8.5.1 Time complexity and practical implementation

The *personalized PageRank/RankProp* method applies the propagation to an entire similarity network. For a network of N nodes this means a time-estimate of $O(iN^2)$ steps where i is the number of iterations. This can be time-consuming for large protein similarity networks that typically have several thousand to several hundred thousand nodes. The situation can be alleviated by considering just the first n objects nearest to the query along with all of their similarities, which leads to a time-estimate of only $O(inN)$, i being the number of iterations [94].

TreeProp-N and *TreeProp-E* apply the propagation to a binary tree. Since constructing the tree with the FastME algorithm requires $O(N^2)$ time in addition to propagation, it is necessary to reduce the size N of the input network. We propose a reduction where we consider just the m nearest neighbors for each of the n top-ranking objects, which will lead to a maximal time estimate of $O(inm)$. A phylogenetic tree is built from nm objects and it will have $nm+1$ leaves and $nm-1$ internal nodes. The time-complexity of propagation by *TreeProp-N* can be estimated as $i(nm+1+3(nm-1)) = O(inm)$, because each internal node of a binary tree has three neighbors. In the case of *TreeProp-E* the computation is similar, except that the internal edges have four neighbours, but the overall time complexity remains $O(inm)$. We point out that *i)* the size of the tree is much fewer than nm since - owing to the clustered nature of the protein similarity space - the lists of the nearest neighbors are largely overlapping; *ii)* the number of iterations is typically less than 20, and *iii)* applied in this way, the algorithms will reorder just the top ranking elements of the original list of similarities. Here, *TreeProp-N* and *TreeProp-E* were written in MatLab using the Bioinformatics Toolbox. For the timing of *RankProp* we ported the original C source code of J. Weston into MatLab. Table 8.1 gives a summary of the approximate time requirements for each algorithm.

Table 8.1: Wall clock time requirements for the *RankProp, TreeProp-N* and *TreeProp-E* algorithms[1]

	RankProp[2]	*TreeProp-N*[3]	*TreeProp-E*[3]
BLAST search[4]	1721.39	1721.39	1721.39
Tree-building	–	1180.13	1180.13
Propagation	15528.2	204.14	205.46
Total:	17249.59	3105.66	3106.98

[1] *Wall clock times were determined on the SCOP40mini database. The preprocessing time necessary to build the network in an all-vs.-all fashion is the same for each algorithm and is not listed here.* [2] *For each protein,* $m = 40$ *nearest neighbors were included in the propagation.* [3] $n = 40$ *top hits and their* $m = 40$ *nearest neighbors were included in the propagation.* [4] *Searching the dataset with one query (this is the same for each algorithm).*

Table 8.1 shows that both tree-based algorithms are faster than the network-based propagation algorithm, and that the gain in time compensates for the extra time-requirement of tree-building. Naturally, network-based propagation (*RankProp*) runs faster if applied to a smaller sized network, but this results in a decrease in performance as mentioned

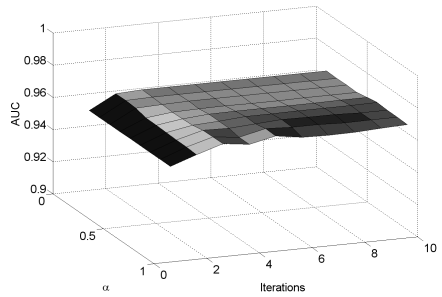


Figure 8.2: Ranking performance of TreeProp-N as a function of the alpha parameter in Eq. 8.4 and the number of iteration steps. The ranking performance is the cumulative ROC AUC value calculated on the 3PGK dataset.

in Section 8.6.

In practical tests, the parameters of *RankProp* were the same as those recommended by Weston et al in [94], i.e. the α parameter was 0.95 and the number of iteration was set to 20. In the case of tree-based methods we used $\alpha = 0.3$ and $i = 20$. These values were chosen because *i)* changing α parameter between $0 - 0.5$ resulted in little variation in performance, and *ii) TreeProp-N* an *TreeProp-E* were typically found to converge in 10 steps or fewer.

The dependence of the performance on α and the number of iterations is shown in Figure 8.2 for *TreeProp-N*. This dependence is quite similar to that found for TreeProp-E (not shown).

8.5.2 Performance evaluation on various databases

The protein datasets were taken from the Protein Classification Benchmark Collection (PCBC, [4]). The 3PGK dataset contains 131 proteins of identical function (id: PCB00016), divided into 10 classification tasks. The SOP40mini dataset contained 1357 proteins grouped by 3D structure (id: PCB00019), divided into 55 classification tasks. The COG dataset contained $17,973$ proteins grouped by function (id: PCB00017) and was divided into 117 classification tasks. From this dataset we evaluated only a few "difficult" tasks in order to test our algorithm.

The algorithms were evaluated in terms of ROC analysis in the way described in Section 3.1. We also calculated the AUC_{50} (ROC_{50}) value for each case [5], because these values are not so sensitive for the class imbalance caused by a large excess of negatives compared to positives, which is a typical situation with all protein datasets analyzed below. In addition to the propagation algorithms (*RankProp, TreeProp-N, TreeProp-E*) we used the simple nearest neighbor evaluation (1NN) as a basis of comparison.

We tested the methods on three protein datasets (3PGK, SCOP40mini, COG) using sequence similarity (BLAST, Smith-Waterman) and/or structural similarity (DALI) [3]. The datasets were selected so as to represent various degrees of difficulty. In the 3PGK dataset, the similarity between group members is high, and between various groups it is also quite high. In SCOP40mini, both the within-group and the between-group

sequence similarities are relatively low. For the COG, the within-group similarities are high and the between-group similarities are low. Since there are several ten to several hundred classification tasks defined on each datasets, we used the cumulative AUC value as a performance indicator. In a few cases (Tables 8.4-8.5) the cumulative AUC was close to 1.00. In these cases we selected a few problematic tasks for comparison.

Tables 8.2-8.3 show the results obtained for sequence comparison methods BLAST and Smith-Waterman. In general, the performance of *TreeProp-N* and *TreeProp-E* are similar to each other and slightly surpass that of *RankProp* followed by 1NN. Out of the 136 cases (for both similarity measure, SW, BLAST), *RankProp* and 1NN were the 'winners' in 28 and 34 cases, respectively.

Table 8.2: Comparison of the performance of algorithms on the 3pgk dataset using the Smith-Waterman scores and BLAST scores[1]

	Smith-Waterman		BLAST	
3pgk	AUC	AUC_{50}	AUC	AUC_{50}
1NN	0.892	0.892	0.899	0.899
RankProp[2]	0.961	0.961	0.963	0.963
TreeProp-N[2]	0.954	0.954	0.951	0.951
TreeProp-E[2]	**0.967**	**0.967**	**0.964**	**0.964**

[1] *The raw scores were used for the comparison.* [2] *The propagation was carried out on the entire dataset, with $i = 20$, $\alpha = 0.95$ iterations for RankProp, and $i = 10$ and $\alpha = 0.3$ for TreeProp-N and TreeProp-E. The whole protein similarity network was used for propagation.*

Table 8.3: The AUC values on the SCOP40mini dataset using the Smith-Waterman scores and BLAST scores[1]

	Smith-Waterman		BLAST	
3pgk	AUC	AUC_{50}	AUC	AUC_{50}
1NN	0.815	0.781	0.763	0.774
RankProp[2]	**0.88**	0.76	0.725	0.655
TreeProp-N[3]	0.86	**0.797**	0.792	**0.808**
TreeProp-E[3]	0.859	0.678	**0.799**	0.754

[1] *The raw scores were used for comparison.* [2] *The $n = 40$ highest similarities were considered for all entries in the database.* [3] *The propagation was carried out for the $n = 40$ top-ranking entries and their $m = 40$ neighbors, in $i = 10$ steps.*

Table 8.5 lists data on a structural comparison for the SCOP40mini dataset. This dataset is difficult to handle complex if we use a sequence comparison (Table 8.3), but it is relatively straightforward if we use an efficient 3D comparison such as DALI. In this case the cumulative AUC was so high that there was hardly any difference between the algorithms, so we chose a few of the most problematic cases for comparison. Even though the AUC values are high, we see the same pattern as we do with sequence comparisons, i.e. in general there is an improvement caused by propagation, and *TreeProp-N* and *TreeProp-E* both perform well compared to *RankProp*.

Table 8.4: Comparison of the performance of algorithms on selected classification tasks defined on the COG dataset, using BLAST scores [1]

COG	Eval.	Classification tasks										
		COG0631	COG0695	COG0697	COG0699	COG0814	COG0842	COG0847	COG1310	COG1752	COG2036	Mean
1-NN	ACU	0.899	0.953	0.985	0.905	0.923	0.952	0.83	0.698	0.948	0.999	0.924
	AUC_{50}	0.977	**1**	0.864	**1**	0.885	0.927	0.98	1	0.991	0.996	0.969
RankProp[2]	ACU	0.862	**0.978**	0.97	0.818	0.912	0.908	0.797	0.47	0.967	1	0.877
	AUC_{50}	0.975	0.978	0.662	0.922	**0.947**	0.693	0.933	**1**	0.898	1	0.829
TreeProp[3]	ACU	0.974	0.95	0.998	0.96	0.91	0.94	0.949	0.978	0.933	1	**0.966**
	AUC_{50}	0.96	0.98	**0.947**	0.996	0.945	**0.977**	**0.997**	0.889	0.98	1	**0.969**
EdgeProp[3]	ACU	**0.976**	0.949	**0.998**	0.886	0.913	0.945	0.954	0.689	0.938	1	0.941
	AUC_{50}	**1**	0.944	0.929	0.996	0.945	**0.977**	**0.997**	0.889	0.98	1	0.969

[1] *The raw scores were used for the comparison.* [2] *The propagation was carried out in the same way as before.* [3] *The propagation was carried out for the* $n = 40$ *top-ranking entries and there* $m = 40$ *neighbors, in* $i = 20$ *steps.*

8.6 Discussion and Conclusions

Phylogenetic trees are used by biologists to highlight the salient internal structure of the protein universe in the form of a "Tree of life". The rationale behind using phylogenetic trees for propagation is based on two assumptions; *i)* Propagation on the salient edges of network may increase the efficiency of the process due to noise reduction; *ii)* tree structures are sparse compared to a full network, so the process will be faster. Here we employed two strategies. TreeProp-N follows the strategy of *PageRank*, the only difference being that the propagation is applied to a tree, rather than to an entire network. TreeProp-E on the other hand propagates the edge-weights to neighboring edges. A further difference with respect to PageRank is the fact that the ranking is carried out according to the distance within a binary tree rather than by using a simple similarity/distance measure.

Performing a ROC analysis on protein datasets (a total of 84 sequence and structure classification tasks) of varying difficulty, we found that *TreeProp-N* and *TreeProp-E* perform somewhat better and are faster than the *personalized PageRank/RankProp* algorithm.

Table 8.5: Comparison of the performance of algorithms on selected classification tasks defined on SCOP40mini dataset, using the DALI 3D-comparison scores.[1]

SCOP40mini	Evaluation	Classification tasks								
		a.39.1._a.39.1.5.	a.4.1._a.4.1.1.	b.1.18._b.1.18.2.	b.1.1._b.1.1.3.	c.2.1._c.2.1.3.	c.37.1._c.37.1.9.	c.47.1._c.47.1.1.	d.81.1._d.81.1.3.	Mean
1-NN	AUC	0.949	0.967	0.988	0.96	0.961	0.903	0.99	0.948	0.958
	AUC_{50}	0.985	0.567	0.857	0.706	0.966	0.735	0.936	0.989	0.843
RankProp[2]	AUC	0.905	0.976	0.978	0.969	0.824	0.961	0.999	0.971	0.948
	AUC_{50}	1	0.685	0.723	0.735	0.98	0.542	0.991	1	0.832
TreeProp[3]	AUC	0.946	1	0.996	0.968	0.997	1	1	0.993	0.987
	AUC_{50}	1	1	0.96	0.883	0.964	1	1	1	**0.976**
EdgeProp[3]	AUC	0.949	1	0.998	0.987	0.993	1	1	0.996	**0.99**
	AUC_{50}	1	1	0.917	0.851	0.976	1	1	1	0.968

[1] *The raw scores were used for for the comparison.* [2] *The propagation was carried out in the same way as before.* [3] *The propagation was carried out for the* $n = 20$ *top-ranking entries and there* $m = 20$ *neighbors, in* $i = 20$ *steps.*

In the practical implementation of *TreeProp-N* and *TreeProp-E*, we used the FastME algorithm [82] for tree construction. We also tested BioNJ [80] and Weighbor [81], which yielded identical results on our datasets, but required more cpu time (data not shown).

For both tree-based algorithms we used a reduced network, namely we selected the n top-ranking objects ($n = 40$) and their $m = 40$ nearest neighbors. Increasing n and m beyond this value did not affect the performance, and values of $n = 20$ and $m = 20$ were found to be generally satisfactory. In principle, this size reduction will make the computation faster, but it can also act as a noise reduction filter, since we select only the "important neighbors" of the query. A question arises of whether the same toplist restriction strategy would improve the performance of *PageRank/RankProp*. However when we used a reduced network (n=m=40) in conjunction with RankProp on the SCOP40 mini dataset (1337 proteins, 55 classification tasks), the AUC value dropped from 0.880 to 0.756. We obtained similar results on the other datasets as well (data not shown) so we believe that the improvements in performance were not induced by the toplist restriction being applied.

Summarizing, we can conclude that tree-structures can be efficiently used to increase the performance of protein classification algorithms. Even though we tested our algorithms on a large variety of classification tests, we think that improvements may critically depend on the type of applications, hence the parameters may need to be adjusted to the datasets being studied.

Chapter 9

The application of ROC analysis for evaluating similarity measures on large-scale class-imbalanced protein datasets

9.1 Introduction

The classification of protein sequences is of fundamental importance in genome annotation so assessing and comparing the efficiency of sequence or structure similarity measures is a crucial task. The method of current choice is ROC (Receiver Operating Characteristic) analysis that evaluates the ranking ability of a similarity measure [103; 104]. Briefly, the members of a database are ranked according to their similarity to a query using a similarity score (such as calculated by BLAST [1], Smith Waterman [2], etc.), and a similarity score is considered efficient, if the proteins belonging to the query's known class (true positives) are on the top of the list. The analysis is carried out by preparing a sensitivity vs. specificity plot, whose integral value, AUC (area under curve) is 1.00 if the true positives are all on the top of the list, and will tend to 0.5 if the ranking is random [103]. In the practice of bioinformatics there are two main variants of this method (Figure 9.2.1). In the element-wise scenario, originally suggested by Gribskov and Robinson [5], one prepares a separate ranked list for all the queries, which will yield one AUC value for each query. The group-wise scenario can be applied if the database is a priori divided into distinct classes, such as domain-types [6; 7]. Here we prepare one single ranking for a protein group, in which we rank the members of the test set with respect to their maximal similarity to the positive train group. Then we calculate a single AUC for the given group. If we wish to calculate a cumulative value for an entire database using either scenario, the usual method is to calculate the arithmetic average of the individual AUC values, what is equivalent to constructing a cumulative AUC plot (such as shown in Figure 9.3 below) and calculating its integral value [6; 7]. In this representation higher curves indicate a better performance. ROC

analysis presupposes the existence of two classes, of which the query's known class is the positive while the rest of the database is the often much larger negative group, in other terms the positive/negative ratio (p/n) is often negligibly small. In order to handle this class imbalance problem and to get a dataset of manageable sizes it is customary to cut the top list at some point, and according to Gribskov and Robinson [5] this can be done by limiting the top list so as to include n negatives (where n is usually taken as some plausible number like 10, 50 etc.). These are the so-called ROC_n (e.g. ROC_{10}, ROC_{50}) values that were originally proposed for the element-wise scenarios but they are also frequently used in the group-wise scenarios. ROC analysis is considered reliable because it includes both specificity and sensitivity, so a method with a high AUC value can be expected to be a robust in a variety of conditions. While we generally agree with this view, we noticed that the ROC_n values can be differentially biased in various classes within a database. The result of this differential bias is that one can not directly compare AUC values between different classes, although AUC values obtained on the same class with different methods are comparable. This is a major problem, since discovery of new protein or gene groups is perhaps the most important tasks in genome research, so there is a need to assess and compare, without class-imbalance artifacts, the predictable aspects of protein groups. No doubt, this assessment could be done by developing and fine-tuning classifier algorithms for each of the new candidate groups, which is, on the other hand too time consuming in view of the amounts of data to be analyzed. Our motivation was to use ROC analysis directly for the purpose. As the positive/negative ratio within a top-list is known to influence the AUC values, we looked for methods where this ratio could be controlled and monitored. Here we propose a simple method where the number of the selected negatives included in the analysis is equal to (or proportional to) the number of positive sample. This balanced ROC analysis provides less biased AUC values, which can be then used to identify difficult groups within a database.

9.2 Materials and Methods

9.2.1 Datasets and algorithm

The SCOP95 dataset used for modelling consisted of protein domain sequences taken from the SCOP database v.1.69, filtered for 95% identity [105]. For comparison we used the Smith-Waterman algorithm [2] as coded in the EMBOSS program package [106] and applied predefined classification tasks for superfamily prediction. The classification tasks and the Smith-Waterman data were taken from record PBS0001 of the Protein Classification Benchmark collection [4], so the positive set contained members of superfamily, one family being the +test set and the rest of the families were incorporated into the +training set. After we performed the evaluation using the element-wise or the group-wise scenario. In each case we used supervised subdivision [8].

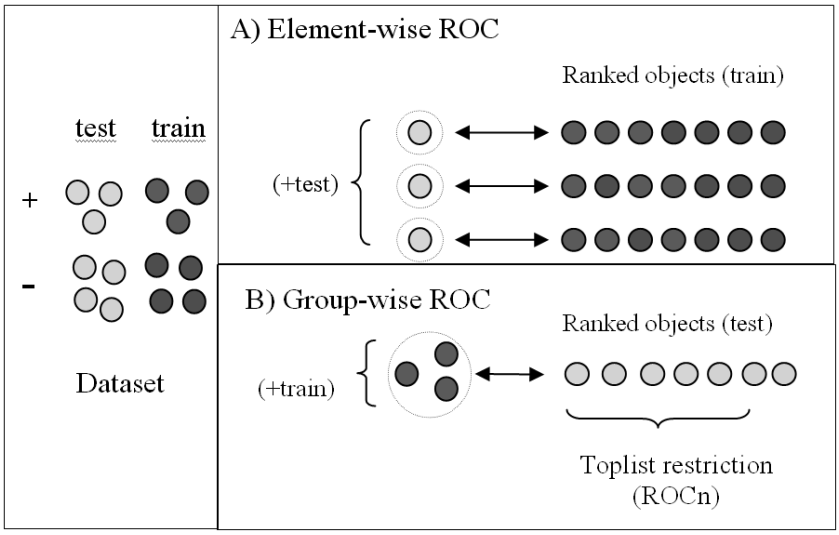


Figure 9.1: In the element-wise scenario (A), each query is compared to a dataset of + and - train examples. A ROC curve is prepared for each query and the integrals (AUC-values) are combined to give the final result for a group of queries. In the group-wise scenario (B) the queries of the test set are ranked according to their similarity to the -train group, and a $ROCAUC$ value calculated from this ranking is used to characterize the group.

9.2.2 ROC calculation

The calculation of AUC was carried out as described [4; 6–8], using top lists including a given number of negative samples. The number of negatives (n) was expressed in multiples of the positive samples (p) participating in the restricted ranking. This normalized number N_r is thus related to the positive/negative ratio, so at $N_r = 1.0$, $p/n \leq 1.0, p/n$ be lower than 1.0 for difficult classes, since in these cases there are positive samples that do not show up in the top lists, especially when less sensitive methods comparison methods are used. For the balanced $ROC(BaROC)$ protocol we selected the top list in such a way that it contained as many negative samples as the number of samples in the entire positive set. This corresponds to $N_r = 1.0$, so the p/n values lie between zero and 1.0. It is equally possible to use a value of $N_r = 2.0$ is to use twice as many negatives as there are positives, in that case the p/n values will be between zero and 0.5. One can easily show that for any N_r, $p/n \leq 1/Nr$. We should add that the BaROC protocol – the same as the ROC_n principle suggested by Gribskov and Robinson [5] – is based on toplist truncation, so the AUC value expected for random ranking is smaller than 0.5.

9.2.3 Likelihood ratio scoring

In the practice of protein sequence classification the input variable of ROC analysis is a sequence similarity score, such as a BLAST or Smith-Waterman score. Recently it has been proposed that a simple likelihood ratio approximant, LR, is a more efficient ranking indicator [107]. The LR score of a query is calculated using the formula

No of negatives in the top list	AUC	p/n
10	0.900	0.10000
50	0.500	0.04000
full negative (5990)	0.833	0.000835

Table 9.1: Dependence of the AUC value on the size of the negative set. Globin-like proteins, $a.1.1.$ in $SCOP95$.

$$LR = \frac{S^+_{max}}{S^-_{max}}, \tag{9.1}$$

where the top similarities are obtained between the query and members of the positive and negative groups, respectively. In contrast to a simple similarity score, LR also contains information on the negative class. A similar scoring measure, an approximant of the posterior probability p is used by the IBk program of the WEKA package [108].

$$p = \frac{S^+_{max}}{S^+_{max} + S^-_{max}} \tag{9.2}$$

Since the likelihood ratio is defined by $p/(1-p)$, it is easy to show that these two indices have in fact the same meaning and thus lead to identical ranking. In our comparisons we used the LR score, along with to simple scoring that uses just. We should point out that LR scoring is meaningful only in the groupwise-scenario, the ranking of the element-wise scenario does not change upon the transformation to LR, so the ranking remains the same as that with simple scoring.

9.3 Results

The heterogeneity of protein groups is a fundamental problem in protein classification. Protein groups vary in terms of the number of group-members, the length and variability of the sequences, the separation from the nearest non-member sequences etc. So we can expect that using a top list of a given length may influence the ROC_n value.

Let us take the globin superfamily $a.1.1$ of the $SCOP95$ data set as an example, and use the Smith-Waterman algorithm to compare sequence similarities. This superfamily has 103 members, and we will define a classification task where family $a.1.1.1$ (Globine-like, 5 members) will be the test group. We will calculate the AUC values in a group-wise scenario, using 10, 50 negatives (which corresponds to the generally used ROC_{10} and ROC_{50} scenarios), or the full dataset applied in the analysis. The results in Table 9.3 indeed show that the values are strongly dependent on the number of negatives taken into the consideration. The phenomenon is shown graphically for the entire $SCOP95$ dataset in which the values represent the average of 246 classification tasks (Figure 9.3). In this case we calculated $ROCAUC$ in three different ways, i.e. using the element-wise scenario, the group-wise scenario with and without likelihood-ratio scoring. The values

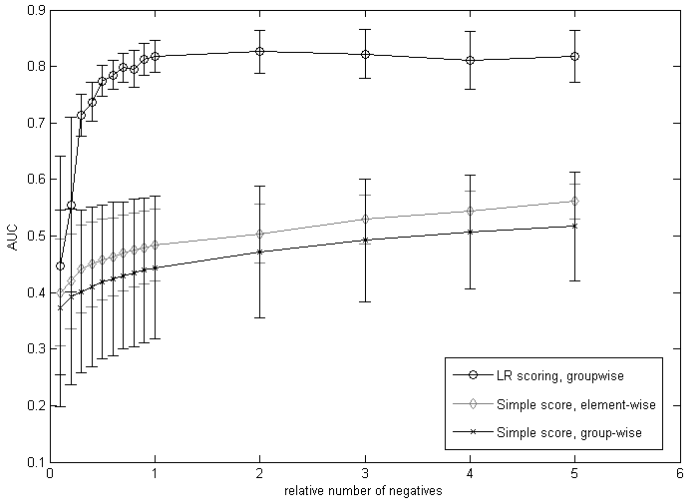


Figure 9.2: Dependence of the $ROCAUC$ values on the size of the negative set. Average AUCs were calculated for all the 246 classifications tasks within the $SCOP95$ dataset. The error bars indicate the average standard deviations. The curves represent different methods of calculation as indicated in the inset and described in Methods. Note that the group-wise scenario with likelihood ratio scoring gives values that are independent of the size of the negative set while the results of the others show an increasing tendency and have higher standard deviation values (indicated by the error bars).

were plotted against the size of the negative set expressed in multiples of the positive set. This was carried out by preparing the ranked list in the usual way and truncating it from the top when the necessary number of negatives was reached. It is conspicuous that the AUC values calculated by any of the three methods show a steeply increasing tendency if the number of negatives is below the size of the positive group. The methods using simple scoring (i.e. when the Smith-Waterman score was used for ROC analysis in the group-wise or in the element-wise scenario) display a increasing tendency above this threshold, but apparent that the curve of likelihood-scoring (in a group-wise scenario) does not, its value being seemingly independent of the number of negative samples in the top list. In addition, the latter method has substantially lower standard deviation values than the other two methods. In principle, a comparison of the scoring methods should not depend on the dataset, and in fact we see that the ranking order of the methods is consistently the same as we reach values of Nr>1. On the other hand, the curves cross each other below Nr=1.00, which shows that the comparison of the methods is not consistent if shorter top lists are used. Taken together, the results confirm the dependence of the values on the number of the negative samples. And since fixing the same size of the negative set for all of the classes will place the different classes at different points within this curve, we can see that there will be a differential bias within the various groups. One of the underlying reasons is that the top lists of a given length may contain a strongly varying number of positives and negatives, so the positive/negative ratio changes as we consider longer and longer top lists. On the other

hand, selecting a number of negatives that is proportional to the size of the positive group will adjust the positive/negative ratio to a level roughly balanced with respect to the group size. With this strategy, the experimenter can be more certain that the AUC differences observed between classes are not class-imbalance artifacts but indicators of quality differences between classes. We propose to use a number of negatives that is equal to or double of the number of the positives participating in the ranking. Figure 9.3 shows that the balanced ROC analysis calculated in this manner is a more stringent test than AUC_{50} or AUC_{10}, at least for the $SCOP95$ dataset in which there are many relatively small groups. The balancing effect is clearly shown in Figure 9.3, where we plotted the AUC value for 246 groups against the p/n ratio. In the case of ROC_{50}, the p/n ratio is way below one (A, left). In the case of the balanced ROC (B, right), p/n gets near one for a number of the groups, but it can be lower in the case of distant similarities since in those cases not all the positives will show up in the top list. In the present dataset we see many such cases since the SCOP dataset is difficult for a sequence comparison method such as Smith Waterman. Simply put, regions in the $BAROC$ scatter diagram allows one to distinguish various classes within a database. A high AUC, high p/n ratio indicates tight groups, with high similarities between the members and well separated from the rest. A high AUC low p/n is characteristic of less well-separated or less tight groups in which the within-group similarity is not sufficiently strong. Finally, low AUC, low p/n ratio is indicative of problem groups that are difficult to predict.

9.4 Discussion

Generally speaking, for a statistical evaluation such as a $ROCAUC$ calculation, it is recommended to have the same number of positive and negative samples be used [103; 104]. This condition is never met in bioinformatics databases, where we have far fewer positives than negatives. A correct solution would be to randomly select equal numbers from the two classes and to construct many classifiers for each case, which is clearly too time consuming given the number of classes and data. Instead, bioinformaticians resort to the truncation of the top lists, which – as we saw in Table 9.3– does not necessarily helps one to decide whether or not a low AUC value is indicative of a problematic object class or it is a class-imbalance artifact. As a way out of this difficulty, we proposed the balanced ROC scenario which adjusts the number of negatives to the level of the size of the positive set. In this manner, classes, where the positives are within the 'balanced' top list, will have a high p/n value as well as a high AUC. These classes are the easy cases for which the comparison measure (in our case the Smith-Waterman similarity score) works efficiently. In the case of difficult classes, we will have fewer positives in the top list. Therefore we can detect problem classes using a scatter plot like the one shown in Figure 9.3, where they will lie in the region of low AUC and low p/n values. In this manner we can make use of the reliability of AUC calculation without having to construct a classifier for all the groups. Likelihood-ratio scoring (Eq. 9.1) provides a further tool for getting rid of the class-imbalance bias, and

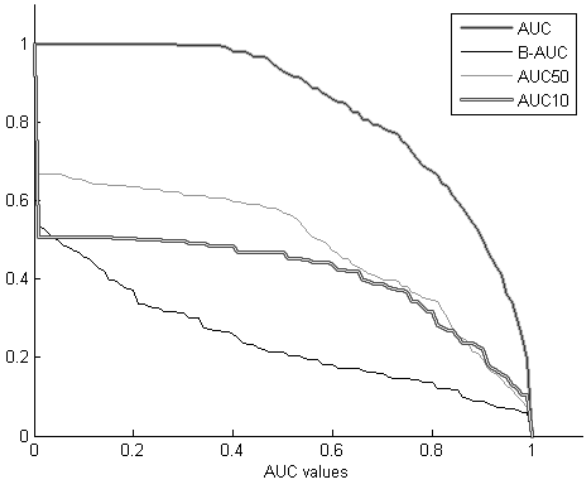


Figure 9.3: Cumulative AUC curves for various calculation/scenarios. The calculations were done on the super families of the SCOP95 database (PCB0001, see Section 9.2.1), using various strategies for top list-restriction. The cumulative AUC curves plot the number of queries or groups (Y-axis) that exceed the AUC value indicated on the X axis. For uniformity, we normalized the Y values to 1.00 by dividing them by the total number of queries or groups, respectively. The AUC value indicates the calculation done on the entire dataset, AUC_{50} and AUC_{10} values indicate calculations based on truncated toplists as suggested by Gribskov and Robinson [5], while the $B-AUC$ value indicates a calculation according to the present $BAROC$ protocol.In this representation the curves with higher values mean a better performance, which in turn means that the AUC on the full dataset gives higher scores than all the other methods, and the balanced ROC gives the lowest scores.

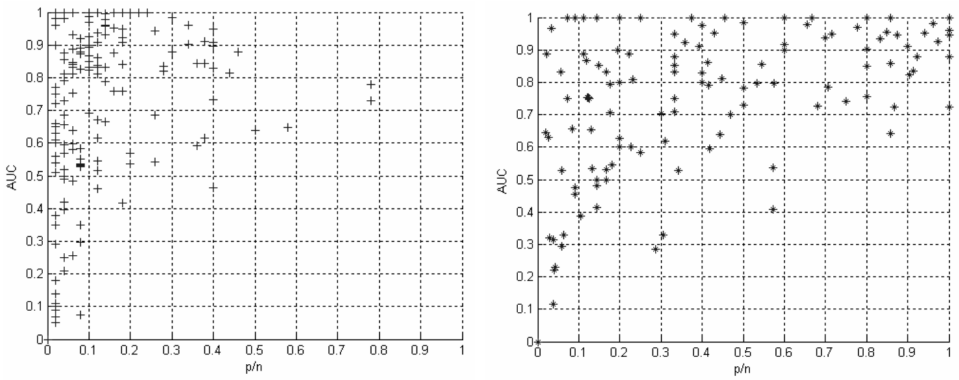


Figure 9.4: The AUC values were calculated for the 246 groups of the SCOP95 database using a group-wise scenario with supervised cross validation, as described [4; 6–8]. The top panel shows the calculation for AUC_{50}, in which the top list of each group contains exactly 50 negative samples. The lower panel shows the balanced ROC, in which the top list contains as many negatives as there are positives in the calculation. $(N_r = 1.00)$. The data points are more spread out.

it can be efficiently used in the BAROC scenario.

Finally, we should mention that the use of the method was illustrated here on 246

sequence classification tasks taken from the SCOP database. We tested this method on the other sequence and 3D classification tasks included in the Protein Classification Benchmark Collection [4], and got identical results (data not shown) which gives us hope that our protocol will be applicable to other classification methods that rank the objects according to a variable which characterizes class-membership.

9.5 Simplified description of the method

We presented a ROC analysis protocol that makes it possible to single out classes in a database that are likely to be difficult to predict. The method, termed Balanced $ROC(BAROC)$, consists in calculating an AUC value for a ranked top list which is truncated so as to contain as many (or twice as many) negative objects as there are positive objects in the entire analysis. In this manner each class will be analyzed with a top list whose length depends on the size of the class. The difficult groups can then be identified by their low AUC values and/or their low positive/negative ratio within the top list. The identification is aided by a scatter plot of AUC vs. positive/negative ratio, as well as by the use of a likelihood ratio-scoring scheme (Eqn. 9.1), that can be efficiently used in the $BAROC$ protocol instead of simple similarity scores.

Chapter 10

Conclusions

For understanding the language of genes and proteins we have to find a suitable model of how they have evolved in the course of evolution. Because of this we need to develop tree building methods which discover the process of evolution. These kinds of methods have gained importance with the advent of molecular biology in the 1970's. Thereafter the implosive advancement in biology allows us to investigate the sequences of the proteins, genes, as well as species/genomes. This is why the microbiological research requires novel and novel phylogenetic analysis tools.

In the first part of this thesis we provided two phylogenetic tree reconstruction methodologies. In the second part, to demonstrate the application of phylogenetic tree reconstruction methods in automatic protein classification, then we introduced protein classification algorithms which make use of phylogenetic tree building methods as well.

The main goal of the first part was to introduce methodologies which can perform a highly accurate phylogenetic analysis. The Multi-Stack algorithm categorically is a distance-based method. Thus it uses only the distance values of the sequences of interest to build a phylogenetic tree. This method is suitable, for example, in constructing a guide tree before multiple alignment.

The second phylogenetic analysis tool was a consensus tree building method, namely the Max Clique Consensus method. It is obvious from the results that the MCC consensus outperforms many widely-used procedures, and it was easy to implement. The time requirement of this method is reasonable (proportional to the tree building method itself), so it can be employed efficiently in a post-processing phase of a phylogenetic analysis tool.

In the second part of this thesis we sought to develop novel and efficient protein classification algorithms. Our basic assumption was that the structure of the biological datasets could be represented by a phylogenetic tree, and using this representation protein classification could be carried out significantly more efficient. This new field of bioinformatics is a very promising area of research.

Appendix A

Summary in English

In this thesis we concentrate on two key topics, namely artificial intelligence and bioinformatics. Within these fields we focus on evolutionary tree reconstruction and machine learning.

Over a hundred years the theory of evolution has became the most acknowledged model of how animal and plant species have developed over time. The discipline which deals with the modelling of evolution is called phylogenetics (the word is originated by the conjunction of the Greek words: phyle = tribe, race and genesis = birth). The methods which are in widespread use in phylogenetics represent the process of species evolution by a so-called phylogenetic tree, which corresponds to a weighted tree-graph where the leaves represent the biological objects of interest. In connection with the reconstruction of these kinds of trees, several problems arise which are interesting from both a computer scientific and a biological point of view.

Earlier phylogenetics focused just on the evolution of species based on morphological characters, but nowadays the explosive advancement in molecular biology also requires the study of proteins. The wealth of sequenced protein data allows us to perform novel investigations. The possibility of comparing protein sequences has moved research work towards the systematization of the proteins isolated from distinct species. Proteins that share a high sequence identity or similarity support the hypothesis that they share a common ancestor, and therefore we call them evolutionary related or homologous proteins. The analysis of evolutionary-related proteins has become a key question in phylogenetics. After our brief introduction we can state the basic goal of the phylogenetics: to reconstruct an appropriate tree topology based on protein sequences which have a high sequence similarity. We should mention here that the high sequence similarity of proteins usually implies that they share common functionality as well, but it does not logically follow.

As the dissertation consists of two parts, the author's results will also be split into two parts. In the *first part*, we introduce then evolutionary tree reconstruction methodologies.

Several tree building method have been worked out and some of them have become widely used, for example the Neighbor Joining (NJ) [9] and the Unweighted Pair Group Method with Arithmetic mean (UPGMA) [10]. These methods belong to the so-called

distance-based or distant matrix methods because they reconstruct the evolutionary history of biological objects based only on pre-determined or observed distance values among them. Our Multi-Stack (MS) [11] algorithm methodologically falls also into this category. Broadly speaking, the MS method finds a weighted tree topology that predicts the observed set of distances as closely as possible. More precisely, a weighted tree defines a distance value for all pair of leaves –i.e. the sum of the weights of edges containing the path between them. Thus the output tree of the MS approach we expect from that the distances defined by itself will differ from the observed distances as small as possible. To find this tree is an NP-complete problem when we have an arbitrary distance measure [12], hence it can only be applied to heuristical solutions. The idea behind the MS method is that it builds the optimal tree for the subsets of the proteins of interest, and then joins these subtrees in an iterative manner. We can apply this bottom-up approach efficiently in many test scenarios, and the MS approach often outperforms many traditional tree building methods.

Since there are many tree building methods which produce more than one possible evolutionary history, or the different tree building methods reconstruct different trees, in many cases it is necessary to use those methodologies which are able to reconstruct one "representative" tree based on several different phylogenetic trees. These kinds of methods are called the consensus tree methods [13], and they are usually applied as the last step of the phylogenetic analysis process.

In general, each inner point in a rooted phylogenetic tree determines a subset of the biological object of interest (i.e. the objects which are represented by those leaves in the tree which lie below the inner point). Exploiting this observation we can see that the concept of a phylogenetic tree and the concept of a hierarchical set system are equivalent. The hierarchical set systems consist of those subsets or, in other words, clusters which are pairwise compatible. Thus each phylogenetic tree corresponds to a pairwise compatible cluster set. Most of the consensus methods determine a compatible subset of the cluster sets of the input trees in different ways, based on the cardinality of clusters' occurrences in the input trees. Their calculations can be done in polynomial time. Our goal is to find the subset of the input clusters for which the total number of the cluster occurrences is maximal. Furthermore, we can also define an arbitrary (not necessarily occurrence-based) weighting function on the clusters of the input trees. We solved this consensus tree building approach efficiently [14], and we showed that it can perform a more precise phylogenetic analysis than the traditional consensus methods (such as the Majority-Rule, the Strict and Greedy consensus methods[15]).

In the *second part* of this thesis we apply the tree building methods in protein classification problems. Automated protein classification is a crucial task in today's biology. The unknown genes/proteins of the distinct organisms can be retrieved and stored in the form of character sequences that are several hundred in length. Nowadays, it has become routine to compare this data to the sequences of known proteins/genes using a method of *approximate string matching*. Then, applying a machine learning method, the unknown protein can be classified into a known category (e.g. structural or functional category) [1]. The automated data annotation system of the frequently

mentioned genome research is based on this methodology.

In this thesis we seek to develop novel and efficient protein classification algorithms. Our basic assumption is that the structure of the biological datasets can be represented by a phylogenetic tree, and using this representation the protein classification can be carried out significantly efficiently [16; 17]. The protein classification methods, which also use phylogenetic information, belong to the field of phylogenomics [18], hence our methods can be viewed as phylogenetic algorithms as well.

A.1 Key Points of the Thesis

In the following a listing of the most important results of the dissertation is given. Table A.1 summarizes which thesis is described in which publication by the author.

I. (a) *The author developed a Multi-Stack based phylogenetic tree building method which makes use of least-squares criteria. In this way he produced a novel algorithm which is competitive with the widely used distance-based tree building methods, and it can reconstruct the evolutionary history of those datasets in a better way where the biological objects (sequences of interest) have lower similarity [11]. This improvement can be shown using evolutionary distances as well as using alignment-free sequence distances. In addition, the MS method achieve a better results in many test scenario than the Fitch-Margoliash algorithm which is also based on the least-squares criteria.*

(b) *The author solved the* Max Clique Consensus *problem via a binary integer programming task. With this approach an arbitrary weighting of subsets one can find the compatible subsets that have maximal weights. In addition, the author introduced a novel Maximum Likelihood weighting scheme, which leads to an efficient phylogenetic reconstruction technique. He tested this method with different evolutionary models and found that this approach in many case outperforms the widely used consensus tree building methods [14]. The trees in the tests were generated by the widely-used PAUP program package[20], and the consensus methods were compared to each other on these trees. Moreover, the author also compared the consensus methods on a real-life database.*

(c) *The author provided a testing framework where the different phylogenetic reconstruction techniques could be compared using different evolutionary models in a wide range [11; 14]. In this testing methodology the biological sequences (DNA or protein) have been generated based on a predetermined model evolutionary tree. Next, on this set of sequences the tree-building method of interest has been applied, and it produces an output tree, which will be compared to the predetermined model tree. Based on the similarity of these trees we can estimate the accuracy of the examined tree reconstruction method. This testing framework provides a more comprehensive testing environment than the bootstrap method [21], because in this framework*

	[11]	[14]	[17]	[16]	[19]
I. (a)	•				
I. (b)		•			
I. (c)	•	•			
II.(a)			•		
II.(b)				•	
II.(c)					•

Table A.1: The relation between the theses and the corresponding publications

we can investigate the efficiency of the tree-building method using different evolutionary models.

II. (a) *The author introduced the TreeInsert and TreeNN methods, which are novel tree-based protein classification algorithms. In contrast to the earlier methods, the algorithms he introduced here make use of just the sequence similarities. Thus they are readily applicable in a wide range on protein classification tasks. The author compared the tree-based methods on many protein classification tasks using ROC analysis, and they were often significant better. The experiments showed that it is worth applying phylogenetic information in protein classification. [17].*

(b) *The author devised two tree-based propagational methods, namely TreeProp-N and TreeProp-E. These methods may be regarded as extensions of TreeNN, because all of these methods update the sequence similarities using the topology of a phylogenetic tree. In experiments these propagational algorithms usually gave a better performance in protein classification comparing to the former systems [16].*

(c) *The author created a ROC analysis-based evaluation method which is a more reliable model evaluation technique than the original ROC analysis when the distribution of the classes is imbalanced. Applying it, a model selection could be carried out more reliably than with the other approaches[19]. He tested this approach on several large-scale datasets.*

Appendix B

Summary in Hungarian

A disszertáció témáját tágabb értelemben a mesterséges intelligencia és a bioinformatika, szorosabb értelemben pedig a gépi tanulás és az evolúciós fák rekonstrukciója képezi.

Az evolúció már több mint egy évszázada a fajok kialakulásának a legelfogadottabb modellje. A törzsfejlődés ezen modellje elsősorban a fajok rokonsági fokát probálja meghatározni. A filogenetika (a szó a görög phülon = törzs és geneszisz = születés szavakból ered) a fajokat, élőlényeket rendszerezi evolúciós rokonsági fokuk alapján. A filogenetikában a legelterjedtebb módszerek a fajok fejlődésének a folyamatát egy úgynevezett filogenetikus fával reprezentálják, amely egy súlyozott fa-gráfnak felel meg, ahol a levelek reprezentálják a vizsgált biológiai objektumokat. Az ilyen típusú fák rekonstrukciója mind biológiai, mind számítástudományi szempontból számos érdekes problémát vet fel.

A különböző fajokból izolált fehérjék szekvenciáinak összehasonlítási lehetősége új típusú vizsgálatok elvégzésére adott alapot a filogenetikában. Ez merőben átformálta a biológia ezen ágát. Míg korábban a filogenetika egyet jelentett a fajok evolúciós fejlődéstanával, addig az új eredmények hatására a kutatások kiterjedtek a fehérjék öröklődésének vizsgálatára. Fehérjék azon csoportját, melyek szekvenciái nagyon hasonlóak egymással, rokon fehérjéknek tekintik, vagy más szóval homológ csoportnak hívjuk. A homológ csoportok általában hasonló funkciókkalrendelkeznek az élő szervezetben. A filogenetika egyik fontos alapfeladatának tekintjük a különböző fajokból izolált, hasonló funkciójú és hasonló szekvenciájú fehérjék vizsgálatát, és ezen fehérjecsoportok evolúciós történetének a meghatározását.

Mivel a disszertáció két fő részre tagolódik, az eredményeket is ennek megfelelően két csoportra fogjuk felosztani.

Az eredmények *első csoportját* filogenetikusfa-építő módszerek bemutatása képezi. A faépítő algoritmusok bemenete sokféle biológiai objektum lehet, úgy mint gén szekvenciák, fehérje szekvenciák vagy mitokondriális DNS szekvenciák egy halmaza. Kimenetük egy fa struktúra, melyben a levelek reprezentálják a vizsgált biológiai objektumokat. Számos faépítő algoritmust dolgoztak ki, amely közül néhány széles körben elterjedt, mint például a Neighbor-Joining [9] és az UPGMA [10]. Ezek a módszerek az úgynevezett távolság-alapú módszerek közé tartoznak, mert a vizsgált szekvenciák előre adott távolságai alapján rekonstruálják az evolúciós történetüket. Ezek a módszerek az evolúciós

történetet általában egy úgynevezett súlyozott filogenetikus fa formájában reprezentálják. Az általunk kidolgozott távolságalapú Multi-Stack (MS) algoritmus [11] azt a súlyozott fatopológiát keresi, amely a legjobban képes visszaadni az előre definiált távolságot: azaz a keresett súlyozott fában a fehérjék távolságai –a közöttük lévő út élsúlyainak az összege– a legkevésbé térnek el az előre definiált távolságoktól. Mivel nem minden esetben létezik olyan súlyozott filogenetikus fa, amely által meghatározott távolságok az előre adott távolságokat teljes mértékben visszaadják, ezért arra törekszünk, hogy a kapott fa topologiája a legjobban igazodjon a "távolságviszonyokhoz". Ennek a fának a megtalálása egy NP-teljes problémára vezet [12], ezért csak heurisztikus megoldást lehet rá adni. Az MS módszer először a vizsgált fehérjék egy-egy részhalmazára épít optimális fát, majd ezeket a részfákat iteratívan összekapcsolja. Ezt a bottom-up megközelítést hatékonyan tudtuk alkalmazni több tesztkörnyezetben, és számos tradicionális faépítőnél jobbnak bizonyult.

Mivel a filogenetikusfa-építő algoritmusok sokszor több lehetséges evolúciós történetet is képesek meghatározni vagy a különböző algoritmusok különböző fát rekonstruálnak, ezért sokszor olyan módszerre van szükségünk a filogenetikus analízis utolsó fázisaként, amely több filogenetikus fa által hordozott információt képes egyetlen reprezentatív fába összegyűjteni [13]. Az ilyen célú algoritmusokat konszenzusfa-építőknek nevezzük. Általában minden gyökeres filogenetikus fa egy belső pontja egyértelműen meghatározza a vizsgált biológiai objektumoknak egy részhalmazát (a belső pont alatt található levelek által reprezentált objektumok halmaza). Tehát a filogenetikus fa ekvivalens a hierarchikus halmazrendszerek vagy más szóval a kompatibilis halmazok konstrukciójával. Ezt a megközelítést alkalmazva, kézenfekvő, hogy azokat a kompatibilis részhalmazokat szeretnénk a konszenzusfa belső pontjaiként kiválasztani, amelyek a vizsgált fákban a legtöbbször fordulnak elő. Természetesen az input fákban előforduló részhalmazokon értelmezhetünk tetszőleges valós értékű súlyfüggvényt, amely nem csupán előforduláson alapszik, hanem az input fák más tulajdonságait is figyelembe veszi. Ezt a konszenzusfa-építési problémát oldottuk meg hatékonyan [14], és megmutattuk, hogy egy alkalmas részhalmaz súlyozással a legelterjedtebb konszenzus módszereknél (mint például Majority-Rule, Strict vagy Greedy konszenzus [15]) pontosabb filogenetikus analízist lehet végrehajtani.

A tézisek *második csoportját* a faépítő módszerek egy alkalmazása képezi. A fehérje-osztályozás az egyik legfontosabb feladat a mai biológiában. Egy-egy szervezet génjeinek adatait szekvenciák –gének által kódolt fehérjéket jelképező néhány száz karakter hosszú sorozatok– formájában tárolják. Mára mindennapi rutinná vált, hogy ezeket az adatokat a közelítő mintaillesztés módszerével összehasonlítják a már ismert fehérjék hasonló adataival, majd valamely osztályozási eljárással megkísérlik besorolni őket a már ismert (szerkezeti, funkciós stb.) kategóriák valamelyikébe [1]. A gyakran emlegetett genom-kutatások automatikus adat-annotációs rendszerei lényegében erre a módszerre épülnek.

Munkáinkban a fehérje-osztályozás újszerű módszereit fejlesztettük ki, melyekben filogentikus információt is használtunk. Alapfeltételezésünk az, hogy a szekvencia adathalmazok belső szerkezete filogenetikus fa formájában ábrázolható, és hogy ennek

révén az osztályozás hatékonnyá tehető [16; 17]. Módszereinkben az ismert és ismeretlen osztállyal rendelkező szekvenciákra megkonstruálunk egy filogenetikus fát csupán a szekvenciák hasonlósági viszonyai alapján. Majd a megkonstruált fából nyerünk ki olyan információt, amely hasznos az osztályozás szempontjából. Azok a fehérjeosztályozási módszerek, amelyekben filogenetikus információt is felhasználnak a filogenomika tárgykörébe tartoznak [18], ezért az általunk kifejlesztett módszerek is ide sorolhatóak.

B.1. Az eredmények tézisszerű összefoglalása

I. (a) *A szerző egy Multi-Stack alapú faépítő módszert dolgozott ki, amely a legkisebb négyzetek kritériumot alkalmazza. Ezáltal egy újszerű faépítő módszert kapunk, amely kompetitív a legelterjedtebb módszerekkel, és pontosabban meg lehet határozni olyan adatbázisok evolúciós történetét, ahol az objektumok hasonlósága alacsonyabb. Ezt a javulást mind illesztés-mentes mind evolúciós távolságok alkalmazásánál ki lehet mutatni. Továbbá a módszer jelentőségét emeli az, hogy a szintén legkisebb négyzetek kritériumot használó Fitch-Margoliash faépítő módszernél[61] számos tesztesetben jobb eredményt ér el a Multi-Stack megközelítés[11].*

(b) *A szerző visszavezette az MCC problémát egy bináris egészértékű programozási feladatra. Ezáltal tetszőleges részhalmazsúlyozás mellett meg lehet határozni a maximális súlyú kompatibilis részhalmazokat. Továbbá a szerző bevezetett egy Maximum Likelihood alapú részhalmaz súlyozást, mely által az MCC hatékonyan alkalmazható konszenzusfa építésre összehasonlítva a legismertebb konszenzusfa-építő módszerekkel. Módszereinket a széles körben elterjedt PAUP programcsomag[20] által konstruált fákon hasonlítottuk össze[14]. Egy valós életből vett fehérjecsoporton bemutattuk a gyakorlati alkalmazhatóságát is.*

(c) *A szerző megadott egy tesztelési keretrendszert, amely alkalmas a faépítő eljárások teljes körű összehasonlítására több evolúciós modell alkalmazásával[11; 14]. A tesztelési módszerben egy előre meghatározott evolúciós fa alapján állítunk elő mesterségesen egy szekvenciahalmazt. Majd ezen szekvenciahalmazra a vizsgált faépítő módszerek alkalmazásával állítjuk elő a filogenetikus fát. Ezek után az eredeti és a kapott fa hasonlósága alapján meg tudjuk becsülni a filogenetikus analízisünk pontosságát. A tesztelési módszer szélesebb körű tesztelést tesz lehetővé, mint a hagyományos bootstrap módszer, mivel a bootstrap módszer egy rögzített szekvenciahalmazból vesz újra mintát [21]. Ezzel szemben ebben a keretrendszerben megvizsgálhatjuk a faépítők viselkedését különböző evolúciós modellek alkalmazása mellett.*

II. (a) *A szerző a tézisében megadja a TreeInsert és a TreNN módszert, melyek újszerű faalapú fehérjeosztályozási eljárások. A korábbi filogenomikai módszerekkel szemben, az itt bemutatott módszerek csak a szekvencia hasonlóságokat használják fel, emiatt egyszerűen alkalmazhatóak széles*

	[11]	[14]	[17]	[16]	[19]
I. (a)	•				
I. (b)		•			
I. (c)	•	•			
II.(a)			•		
II.(b)				•	
II.(c)					•

B.1. táblázat. A tézispontok és a Szerző publikációinak viszonya

körben. Több fehérje osztályozási problémán összehasonlította a faalapú módszereket ROC analízis alkalmazásával, és jelentős javulást ért [17]. Az eredmények rámutatnak, hogy érdemes filogenetikus információt alkalmazni fehérje osztályozásban.

(b) *A tézisben kidolgozásra került két filogenetikusfa-alapú propagációs módszer, a TreeProp-N és a TreeProp-E. Ezek a módszerek a TreeNN algoritmus kiterjesztéseinek tekinthetőek olyan módon, hogy a filognetikus fa struktúráját felhasználva a szekvencia hasonlóságokat felüldefiniálják. Ezen propagációs módszerek fehérjeosztályozásban további javulást eredményeztek a korábbi módszerekhez képest. [16].*

(c) *A szerző definiált egy ROC analízisen alapuló kiértékelési módszert, mellyel egy megbízhatóbb mérőszám kapható a szekvenciahasonlóság minőségére abban az esetben, ha kiegyensúlyozatlan az osztályok eloszlása az adatbázisban [19]. Ezáltal sokkal megbízhatóbb modellkiértékelést lehet végrehajtani a ROC analízis segítségével. Az itt bevezetett módszert a szerző nagy méretű fehérjeadatázison tesztelte.*

Bibliography

[1] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J Mol Biol*, 215(3):403–410, October 1990.

[2] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, March 1981.

[3] L. Holm and C. Sander. Dali: a network tool for protein structure comparison. *Trends Biochem Sci.*, 11(20):478–80, 1995.

[4] P. Sonego, M. Pacurar, S. Dhir, A. Kertész-Farkas, A. Kocsor, Z. Gáspari, A. M. Leunissen, J., and S. Pongor. A protein classification benchmark collection for machine learning. *Nucleic Acids Research*, 35(Supplement 1):D232–D236, January 2007.

[5] M. Gribskov and N. Robinson. Use of receiver operating characteristic (roc) analysis to evaluate sequence matching, 1996.

[6] T. Jaakkola, M. Diekhans, and D. Haussler. A discriminative framework for detecting remote protein homologies. *Journal of Computational Biology*, 7(1-2):95–114, 2000.

[7] L. Li and W. Noble. Combining pairwise sequence similarity and support vector machines for remote protein homology detection, 2002.

[8] A. Kertész-Farkas, S. Dhir, P. Sonego, M. Pacurar, S. Netotea, H. Mijveen, A. Kuzniar, J.A.M. Leunissen, A. Kocsor, and S. Pongor. Benchmarking protein classification algorithms by supervised crossvalidation. *Journal of Biochemical and Biophysical Methods*, page doi:10.1016/j.jbbm.2007.05.011, 2007.

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

[10] F. J. Rohlf. Classification of aedes by numerical taxonomic methods (diptera: Culicidae). *Ann Entomol Soc Am*, 56:798–804, 1963.

[11] R. Busa-Fekete, A. Kocsor, and Cs. Bagyinka. A multi-stack based phylogenetic tree building method. *Lencture Notes in Bioinformatics*, 4463:49–60, 2007.

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

[13] E.N. Adams. Consensus techniques and the comparison of taxonomic trees. *Systematic Zoology*, 21:390–397, 1972.

[14] R. Busa-Fekete, A. Bánhalmi, A. Kocsor, and Cs Bagyinka. A binary integer programming relaxation for the max clique consensus. *submitted*, 2008.

[15] D. Bryant. A classification of consensus methods for phylogenetics. *Bioconsensus,Discrete Mathematics and Theoretical Computer Science*, 61:163–184, 2001.

[16] A. Kocsor, R. Busa-Fekete, and S. Pongor. Protein classification based on propagation on unrooted binary trees. *Protein and Peptide Letters*, page in press, 2008.

[17] R. Busa-Fekete, A. Kocsor, and S. Pongor. Tree-based algorithms for protein classification. In *Computational Intelligence in Bioinformatics*, pages 165–182. 2008.

[18] J.A. Eisen. Phylogenomics: improving functional predictions for uncharacterized genes by evolutionary analysis. *Genome Res.*, 8:163–7, 1998.

[19] R. Busa-Fekete, A. Kertész-Farkas, A. Kocsor, and S. Pongor. Balanced roc analysis (baroc) protocol for the evaluation of protein similarities. *Journal of Biochemical and Biophysical Methods*, page doi:10.1016/j.jbbm.2007.06.003, 2007.

[20] D. Swofford. Paup program package. *http://paup.csit.fsu.edu/index.html*, 2007.

[21] J. Felsenstein. *Inferring Phylogeneties*. Sinauer, 2004.

[22] C. Semple and M. Steel. *Phylogenetics*. Oxford University Press, 2003.

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

[24] M. Kuhner and J. Felsenstein. A simulation comparison of phylogeny algorithms under equal and unequal evolutionary rates [published erratum appears in mol biol evol 1995 may;12(3):525]. *Mol Biol Evol*, 11(3):459–468, 1994.

[25] C. M. Zmasek and S. R. Eddy. A simple algorithm to infer gene duplication and speciation events on a gene tree. *Bioinformatics*, 17(9):821–828, September 2001.

[26] J. Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *J Mol Evol*, 17(6):368–376, 1981.

[27] A. W. F. Edwards and L. L. Cavalli-Sforza. Reconstruction of evolutionary trees. *Annals of Human Genetics*, 27:105–106, 1963.

[28] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A*, 89(22):10915–10919, 1992.

[29] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *J Comput Biol*, 1(4):337–348, 1994.

[30] J.D. Thompson, D.G. Higgins, and T.J. Gibson. Clustal w: improving the sensitivity of progressivemultiple sequence alignment through sequence weighting,position-specific gap penalties and weight matrix choice. *Nucleic Acids Res.*, 22:4673–4680, 1994.

[31] S. B. Needleman and C. D. Wunsch. 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, March 1970.

[32] S. F. Altschul, T. L. Madden, A. A. Schäffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402, September 1997.

[33] T Müller and M. Vingron. Modeling amino acid replacement. *J. Comp. Biol.*, 6:761–776, 2000.

[34] J. Felsenstein. *Evolution*, 38:16–24, 1984.

[35] M. Hasegawa, H. Kishino, and T. Yano. Dating of the human-ape splitting by a molecular clock of mitochondrial dna. *J. Mol. Evol.*, 22(2):160–74, 1985.

[36] R. E. Dickerson. The structures of cytochrome c and the rates of molecular evolution. *J. Mol. Evol.*, 1:26–45, 1971.

[37] S. Vinga and J. Almeida. Alignment-free sequence comparison-a review. *Bioinformatics*, 19(4):513–523, March 2003.

[38] R. Cilibrasi and P. Vitányi. Clustering by compression. *IEEE Transactions on Infomation Theory*, 2004.

[39] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Infomation Theory*, IT-23:337–343, 1977.

[40] C. G. Nevill-Manning and I. H. Witten. Compression and explanation using hierarchical grammars. *Computer Journal*, 4(2/3):103–116, 1997.

[41] D. Bryant and P. Waddell. Rapid evaluation of least-squares and minimum-evolution criteria on phylogenetic trees. *Journal of Biochemical and Biophysical Methods*, 15(10):1346–1359, 1998.

[42] A. Rzhetsky and M. Nei. Theoretical foundation of the minimum-evolution method of phylogenetic inference. *Mol. Biol. Evol.*, 10:1073–1095, 1993.

[43] W. M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155(760):279–284, January 1967.

[44] L.R. Foulds and R.L. Graham. *Advances in Applied Mathematics*, (6):43–49, 1982.

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

[46] K. Atteson. The performance of neighbor-joining methods of phylogenetic reconstruction. *Algorithmica*, 25, 1999.

[47] D. Bryant and P. Waddell. Rapid evaluation of least-squares and minimum-evolution criteria on phylogenetic trees. *Journal of Biochemical and Biophysical Methods*, 15(10):1346–1359, 1998.

[48] L. Cavalli-Sforza and A. Edwards. Phylogenetic analysis models and estimation procedures. *Evolution*, 32:550–570, 1967.

[49] Levenberg-Marquardt nonlinear least squares algorithms in C/C++. Manual. *http://www.ics.forth.gr/~lourakis/levmar/*.

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

[51] H. Matsuda. Protein phylogenetic inference using maximum likelihood with genetic algorithm. In *Pacific Symposium on Biocomputing*, pages 512–523, 1996.

[52] P. A. Goloboff. Analyzing large data sets in reasonable times: Solutions for composite optima. *Cladistics*, 15(4):415–428, 1999.

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

[54] Gopalakrishnan P.S. Bahl, L.R. and R.L. Mercer. Search issues in large vocabulary speech recognition. In *Proceedings of the 1993 IEEE Workshop on Automatic Speech Recognition, Snowbird, UT.*, 1993.

[55] G. Gosztolya and A. Kocsor. Improving the multi-stack decoding algorithm in a segment-based speech recognizer. In *IEA/AIE*, pages 744–749, 2003.

[56] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications (Texts in Computer Science)*. Springer, February 1997.

[57] T. H. Jukes and C. R. Cantor. Evolution of protein molecules. *Mammalian Protein Metabolism, Academic Press, New York*, edited by H. N. MUNRO:21–132, 1969.

[58] E. Zuckerkandl and L.B. Pauling. Molecular disease, evolution, and genetic heterogeneity. *Horizons in Biochemistry.*, pages 189–225, 1962.

[59] I. V. Ovchinnikov, A. Götherström, G. P. Romanova, V. M. Kharitonov, K. Lidén, and W. Goodwin. Molecular analysis of neanderthal dna from the northern caucasus. *Nature*, 404(6777):490–493, March 2000.

[60] P. M. Vignais, B. Billoud, and J. Meyer. Classification and phylogeny of hydrogenases. *FEMS Microbiology Reviews*, 25:455–501, 2001.

[61] J. Felsenstein. Phylip program package. *http://evolution.genetics.washington.edu/phylip.html*, 2007.

[62] C. A. Phillips and T. Warnow. The asymmetric median tree: a new model for building consensus trees. *Discrete Applied Mathematics, Special Issue on Computational Molecular Biology*, pages 311–335, 1996.

[63] D. Bryant. *Hunting for trees, building trees and comparing trees: theory and method in phylogenetic analysis*. PhD thesis, Dept. Mathematics, University of Canterbury, 1997.

[64] A.H. Land and A.G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, 28:497–520, 1960.

[65] J. Egerváry. On combinatorical properties of matrices. *translated by H.W. Kuhn., Logistic Papers*, 11:1–11, 1931.

[66] Mathworks Inc. Matlab. *http://www.mathworks.html*, R2006a.

[67] MOSEK Optimization Software. Manual. *http://www.mosek.com*, 4.0.0.60.

[68] L.S. Jermiin, G.J. Olsen, K.L. Mengerson, and S. Easteal. Majority-rule consensus of phylogenetic trees obtained by maximum-likelihood analysis. *Mol. Biol. Evol.*, 14:1296–1302, 1997.

[69] G. Yule. A mathematical theory of evolution. *Based on the conclusions of Dr. J. C. Willis. Philos. Trans. Roy. Soc. London Ser. B, Biological Sciences*, 213:21–87, 1925.

[70] F. R. McMorris and R. C. Powers. Consensus weak hierarchies. *Bulletin of Mathematical Biology*, 53:679–684, 1991.

[71] A. Rambaut and N. C. Grassly. Seq-gen: an application for the monte carlo simulation of dna sequence evolution along phylogenetic trees. *Comput. Appl. Biosci.*, 13:235–238, 1997.

[72] M. Kimura. A simple method for estimating evolutionary rate of base substitutions through comparative studies of nucleotide sequences. *Journal of Molecular Evolution.*, 16:111–120, 1980.

[73] L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *Ann. Math. Stat.*, 37:1554–1563, 1966.

[74] H. A. Schmidt, K. Strimmer, M. Vingron, and A. von Haeseler. Tree-puzzle: maximum likelihood phylogenetic analysis using quartets and parallel computing. *Bioinformatics*, 18:502–504, 2002.

[75] J. P. Huelsenbeck and F. Ronquist. Mrbayes: Bayesian inference of phylogenetic trees. *Bioinformatics*, 17(8):754–755, 2001.

[76] K. Sjölander. Phylogenomic inference of protein molecular function: advances and challenges. *Bioinformatics*, 20(2):170–179, January 2004.

[77] D. M. Cuturi and J. P. Vert. The context tree kernel for strings. *Neural Networks*, 18:1111 – 1123, 2004.

[78] G. Szarvas, R. Farkas, and R. Busa-Fekete. State-of-the-art anonymization of medical records using an iterative machine learning framework. *J Am Med Inform Assoc.*, 14, 2007.

[79] H. Saigo, J.P. Vert, N. Ueda, and T. Akutsu. Protein homology detection using string alignment kernels. *Bioinformatics*, 20:1682–1689, 2004.

[80] O. Gascuel. Bionj: an improved version of the nj algorithm based on a simple model of sequence data. *Mol Biol Evol*, 14(7):685–695, July 1997.

[81] J. B. William, D. S. Nicholas, and L. H. Aaron. Weighted neighbor joining: A likelihood-based approach to distance-based phylogeny reconstruction. *Mol. Biol. Evol.*, 1(17):189–197, 2000.

[82] R. Desper and O. Gascuel. Fast and accurate phylogeny reconstruction algorithms based on the minimum-evolution principle. *Journal of Computational Biology*, 5(9):687–705, 2002.

[83] P. Baldi, S. Brunak, Y. Chauvin, C. A. Andersen, and H. Nielsen. Assessing the accuracy of prediction algorithms for classification: an overview. *Bioinformatics*, 16(5):412–424, May 2000.

[84] V. B. Bajic. Comparing the success of different prediction software in sequence analysis: a review. *Briefings in Bioinformatics*, 3:214–28, 2000.

[85] P Flach and S. Wu. Repairing concavities in roc curves. In *Proc. 2003 UK Workshop on Computational Intelligence.*, pages 38–44, 2003.

[86] P. Larrañaga, B. Calvo, R. Santana, C. Bielza, J. Galdiano, I. Inza, J. A. Lozano, R. Armañanzas, G. Santafé, A. Pérez, and V. Robles. Machine learning in bioinformatics. *Brief Bioinform*, 7(1):86–112, March 2006.

[87] R. L. Tatusov, N. D. Fedorova, J. D. Jackson, A. R. Jacobs, B. Kiryutin, E. V. Koonin, D. M. Krylov, R. Mazumder, S. L. Mekhedov, A. N. Nikolskaya, B. S. Rao, S. Smirnov, A. V. Sverdlov, S. Vasudevan, Y. I. Wolf, J. J. Yin, and D. A. Natale. The cog database: an updated version includes eukaryotes. *BMC Bioinformatics*, 4, September 2003.

[88] B. Lazareva-Ulitsky, K. Diemer, and P. D. Thomas. On the quality of tree-based protein classification. *Bioinformatics*, 21(9):1876–1890, May 2005.

[89] Li Q. Pollack, J.D. and D.K. Pearl. Taxonomic utility of a phylogenetic analysis of phosphoglycerate kinase proteins of archaea, bacteria, and eukaryota: insights by bayesian analyses. *Mol Phylogenet Evol*, 35:420–430, 2005.

[90] S. L. Lauritzen and D. J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems.

[91] J. Pearl. *Probabilistic Reasoning in Intelligent Systems : Networks of Plausible Inference*. Morgan Kaufmann, September 1988.

[92] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, April 1998.

[93] R. Motwani and P. Raghavan. *Randomized Algorithms (Cambridge International Series on Parallel Computation)*. Cambridge University Press, August 1995.

[94] J. Weston, A. Elisseeff, D. Zhou, C. S. Leslie, and W. S. Noble. Protein ranking: from local to global structure in the protein similarity network. *Proc Natl Acad Sci U S A*, 101(17):6559–6563, April 2004.

[95] R. Kuang, J. Weston, W. S. Noble, and C. Leslie. Motif-based protein ranking by network propagation. *Bioinformatics*, 21(19):3711–3718, October 2005.

[96] M. Leone and A. Pagnani. Predicting protein functions with message passing algorithms. *Bioinformatics*, 21(2):239+.

[97] D. Zhou, J. Weston, A. Gretton, O. Bousquet, and B. Schölkopf. Ranking on data manifolds.

[98] H. Hegyi and S. Pongor. Predicting potential domain-homologies from fasta search results. *Bioinformatics(CABIOS)*, 3(9):371–372, 1993.

[99] J. Murvai, K. Vlahovicek, E. Barta, S. Parthasaraty, H. Hegyi, F. Pfeiffer, and S. Pongor. The domain-server: direct prediction of protein domain-homologies from blast search. *Bioinformatics*, 4(15):343–344, 1999.

[100] S. Hassan and C. Banea. Random-walk termweighting for improved text classification. pages 53–60, 2006.

[101] R. Mihalcea, P. Tarau, and E. Figa. Pagerank on semantic networks, with application to word sense disambiguation. In *Proceedings of The 20st International Conference on Computational Linguistics*, page Article No. 1126, 2004.

[102] B.N. Parlet. *The symmetric eigenvalue problem*. Prentice-Hall,Englewood, Cliffs, NJ, 1994.

[103] J.P. Egan. *Signal Detection theory and ROC Analysis*. New York: Academic Press, 1975.

[104] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, November 2000.

[105] A. Andreeva, D. Howorth, S. E. Brenner, T. J. Hubbard, C. Chothia, and A. G. Murzin. Scop database in 2004: refinements integrate structure and sequence family data. *Nucleic Acids Res*, 32(Database issue), January 2004.

[106] P. Rice, I. Longden, and A. Bleasby. Emboss: The european molecular biology open software suite. *Trends in Genetics*, 16(6):276–277, June 2000.

[107] Kajan L., Kertesz-Farkas A., Franklin D., Ivanova N., Kocsor A., and Pongor S. Application of a simple likelihood ratio approximant to protein sequence classification. *Bioinformatics*, 22(23):2865–2869, December 2006.

[108] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, second edition, June 2005.