Approaches to Efficient Multiple Sequence Alignment and Protein Search

Adrienn Szabó

Supervisor: István Miklós

Eötvös Loránd University, Budapest

Faculty of Informatics

PhD School of Computer Science

Erzsébet Csuha-Várjú, D.Sc.

PhD Program of ”Basics and methodology of informatics”

János Demetrovics, D.Sc.

A dissertation submitted for the degree of

Philosophia Doctor (PhD)

2016
Acknowledgements

Firstly, I would like to thank the extensive help and guidance of my supervisors, István Miklós and András A. Benczúr. I would also like to express my gratitude to Máté R. Bachorecz and Nóra Juhász, who helped tremendously in developing, coding and testing the software packages of Chapter 5.

Lastly, I list here all my friends and colleagues at MTA SZTAKI, who contributed to the final version of this thesis by pointing out several typos and mistakes: Ákos Csaba, Dávid Balambér, Tamás Kiss, Júlia Pap, Dávid Siklósi, Péter Szabó.
## Contents

1 Introduction
   1.1 Data mining ........................................... 2  
   1.2 Bioinformatics ....................................... 3

2 Sequence alignment methods .................................. 5  
   2.1 Theory and practice of sequence alignment ............ 6  
   2.2 Corner cutting methods ................................ 32  
   2.3 Progressive alignment .................................. 35  
   2.4 Software tools for MSA ................................. 38  
   2.5 Handling alignment uncertainty ........................ 41  
   2.6 Introduction to protein search methods .............. 43

3 Reticular Alignment ........................................... 47  
   3.1 Introduction ........................................... 47  
   3.2 Methods ................................................. 50  
   3.3 Results and discussion .................................. 59  
   3.4 Implementation details and authors’ contributions .... 65

4 Efficient representation of uncertainty in MSA .......... 69  
   4.1 Introduction ........................................... 69  
   4.2 Methods ................................................. 70
Introduction

In this dissertation I will focus on two topics: alignment of biological sequences, and searching protein families in a large database. The latter is more close to a traditional data mining task performed over biological data, while sequence alignment is a very challenging and active research subfield within bioinformatics in itself.

Both of my main topics are related to the “trendy” research fields of *Data Mining* and *Bioinformatics*. I will briefly introduce these broad topics here, in Chapter 1.

Then, in Chapter 2, I will go into more detail about previously existing solutions and techniques by reviewing the related scientific literature.

Chapter 3 and Chapter 4 are about developing new and efficient solutions regarding protein alignment and dealing with the inherent uncertainty after an alignment (or many of them) have been found.

Chapter 5 shows how I tackled the problem of finding a needle in a haystack within a biological context: the task was to find protein families exhibiting some pre-defined complex features in a large protein database.

Finally, Chapter 6 Summarizes the results and draws the conclusions.
1. INTRODUCTION

1.1 Data mining

Data mining has attracted a great deal of attention in the information industry and in society as a whole in recent years, due to the wide availability of huge amounts of data and the need for turning such data into useful information and knowledge [1].

In the Foreword of “The” data mining book of Jiawei Han and Micheline Kamber, Christos Faloutsos writes:

“Every enterprise benefits from collecting and analyzing its data: Hospitals can spot trends and anomalies in their patient records, search engines can do better ranking and ad placement, and environmental and public health agencies can spot patterns and abnormalities in their data. The list continues, with cybersecurity and computer network intrusion detection; monitoring of the energy consumption of household appliances; pattern analysis in bioinformatics and pharmaceutical data; financial and business intelligence data; spotting trends in blogs, Twitter, and many more. Storage is inexpensive and getting even less so, as are data sensors. Thus, collecting and storing data is easier than ever before.”

Of course, collecting the data is far from enough; after cleaning and tidying the raw dataset, we need intelligent algorithms to find the useful gems of information in the deluge of data.

Usually the term knowledge discovery from data, or KDD is used interchangeably with data mining, but sometimes data mining is treated as only one step within the KDD process.

This process generally consists of the following steps [1]:

1. Data cleaning (to remove noise and inconsistent data)

2. Data integration (where multiple data sources may be combined)
3. Data selection (where data relevant to the analysis task are retrieved from the database)

4. Data transformation (where data are transformed and consolidated into forms appropriate for mining by performing summary or aggregation operations)

5. Data mining (an essential process, where intelligent methods are applied to extract data patterns)

6. Pattern evaluation

7. Knowledge presentation (where visualization and knowledge representation techniques are used to present mined knowledge to users)

Data mining has always been an interdisciplinary field – in the intersection of computer science, database management, artificial intelligence and data visualization – but as the amount of measured biological data is increasing, there is an obvious need for more specific solutions that effectively help the work and research of biologists.

We will see an example of a data mining pipeline carried out on biological (protein) data in Chapter 5.

1.2 Bioinformatics

In the beginning of the 1970s, Ben Hesper and Paulien Hogeweg started to use the term “bioinformatics” for the research they wanted to do, defining it as “the study of informatic processes in biotic systems”.

They realized quite early that one of the most important defining properties of life was information processing in its various forms (e.g., information accumulation during evolution and information transmission from DNA to intra- and
intercellular processes), and that interpretation of such information at multiple levels is essential if we would like to understand how life on Earth works.

Propelled by the exponential increase of sequence data, the term bioinformatics became mainstream in the late 1980s, coming to mean the development and use of computational methods for data management and data analysis of sequence data, protein structure determination, homology-based function prediction, and phylogeny [2].

Now, in the first decades of the 21st century, the term bioinformatics means a broad interdisciplinary field, having roots in computer science, statistics, mathematics and engineering, that aims to study, process, and visualize biological data. As the computational power of our machines ever grows, we have to face the challenge of finding a delicate balance between a model’s strength in describing the real world accurately and the required running times / memory consumption again and again.

In this thesis I will focus on two closely related areas of bioinformatics: *sequence alignment* and *searching proteins* with a pre-defined pattern in a large database.
Sequence alignment methods

In this chapter I will give an overview of the literature and existing methods related to the new findings that will be discussed in chapters 3, 4, and 5.

When we compare the sequences of genes and proteins, the similarities and differences – at the level of individual bases or amino acids – are analyzed, with the aim of inferring structural, functional, and evolutionary relationships among the sequences under study. The most common comparative method is sequence alignment, which provides an explicit mapping between the residues of two or more sequences [3].

The simplest case is when our goal is to determine similarities between only two sequences. I will discuss methods and techniques for pairwise alignment in subsection 2.1.4.

Multiple sequence alignment (MSA) – where we compare three sequences at least – has long been regarded as the Holy Grail of bioinformatics, for two reasons: its importance as a tool for protein structure and function prediction, phylogeny inference and other common tasks in sequence analysis; and because it turns out to be a computationally hard problem. MSA plays an essential role in two related areas of molecular biology: finding highly conserved subregions
among a set of biological sequences, and inferring the evolutionary history of some species from their associated sequences [4, 5]. I will go into details in section 2.1.5.

2.1 Theory and practice of sequence alignment

During mitosis – the ordinary cell division – the DNA is duplicated within a cell, then it is divided into two parts so that the two daughter cells each have the same set of genetic information as the parent cell had. Meiosis, on the other hand, produces four daughter cells – usually gametes or spores, for sexual reproduction – with halved amounts of genetic information. The replication of DNA strands is the key element of these cell division types. This process is both impressively fast and accurate, although not perfect: about 1 mistake is made for every $10^8$ nucleotides copied [6]. These mutations are mostly lethal or detrimental to the cell, but some are not; in the long term they enable adaptation of species to a changing environment, and – in the time span of thousands to millions of years – the appearance of new species.

When trying to align two related sequences – which essentially means matching the characters (denoting the four bases of a DNA sequence or 20 different kinds of amino acids that make up a protein) of the two strings of biological symbols – there are three different ways of how things can “go wrong”. The three simple types of mutation because of which we can not match exactly the consecutive symbols of the strings are:

Substitution is an exchange between two single chemical letters, i.e. nucleotides or bases in the DNA, such as switching a T (thymine) to a C (cytosine); or changing a single amino acid, for example from serine (S)
to proline (P) in the protein chain. ¹

**Insertion** is a mutation in which extra nucleotides are inserted between two symbols of the original DNA string, or, consequently, new amino acids are inserted into the protein.²

**Deletion** happens when a section of DNA or the protein is lost, or deleted.

More complex kinds of mutations (inversion, duplication, translocation) exist, but the algorithms discussed in this thesis do not deal with these explicitly so I do not define them.

As a small example, let us say we would like to align two short DNA sequences, named *seq1* and *seq2*:

<table>
<thead>
<tr>
<th>seq1</th>
<th>AACAGGAT</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq2</td>
<td>ATCGATT</td>
</tr>
</tbody>
</table>

An alignment of these could look like (by convention, the upper sequence is the hypothesized ancestor, or, the more archaic sequence):

<table>
<thead>
<tr>
<th>seq1</th>
<th>AACAGGAT-</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq2</td>
<td>ATC-G-ATT</td>
</tr>
</tbody>
</table>

At positions 1, 3, 5, 7, and 8 we see *matches* – the upper and lower characters are the same in these columns. Position 2 shows a *mismatch* where an A has been transformed into a T. At positions 4 and 6 there are deletions, marked by dashes; and the last column shows an insertion of a T. Insertions and deletions are the two sub-kinds of *gaps*, usually denoted by dashes.

¹It is worth noting that sometimes substitutions within the DNA may not effect the protein structure, such mutations are called “silent mutations”. On the other hand, there are examples where even one small change like mentioned above (also called an SNP, as *Single Nucleotide Polymorphism*) can cause serious effects: sickle cell anemia is caused by a substitution in the beta-hemoglobin gene, which alters a single amino acid in the protein produced.

²Because of the fact that three bases (a *triplet*) of the DNA encode together one amino acid of the produced protein, insertions (or deletions) into the DNA of lengths not divisible by three almost always ruin the encoded protein’s function entirely.
2. SEQUENCE ALIGNMENT METHODS

The choice of aligning the character G of the second sequence to the first one of the two adjacent G-s within the ancestor string was now arbitrary. If we take into account the general observation that the number of separate gap-sequences tends to be low in manually created, accepted alignments (the underlying biological reason being that the probability of a longer insertion or deletion is not much lower than in the case of a shorter one; but many short indels\(^1\) are less probable), then we might come up with an even “nicer” alignment:

```
| seq1 | AACAGGAT-   |
| seq2 | ATC--GATT   |
```

The number of mismatches and indel characters is exactly the same in the previous two mini-alignments, but still the second feels to be a better one. The reason is not mathematical, but has biological origin. The difference is: there is only one gap (although a longer one) in the ancestor sequence of the second alignment, whereas in the first alignment, there are two separate (shorter) gaps. The two separate gaps could have appeared with less probability if we believe in a model (underpinned by data) in which the length of a gap matters less, so an opening of a new gap is penalized more than extending a gap. We will explore gap penalty scoring systems in subsection 2.1.2, some of them will address this problem.

In the following parts I will focus mainly on aligning protein sequences, although most methods would work on DNA / RNA sequences as well (with other parameters when applicable).

2.1.1 Substitution matrices

Whenever we have to match a character with a non-equivalent other character during the alignment process, we have to define the “badness” of the mismatch. A very simplistic approach could be to give the same positive score to each

\(^1\) *indel* is a frequently used common short name for insertions or deletions
2.1 Theory and practice of sequence alignment

matching character pair, and the same negative score to the non-matching pairs of an alignment. Even this simple scoring scheme can give reasonable results if the match / mismatch score ratios are selected according to our expectations about the similarity of the sequences under study. A simple statistical model – as explained for example in [7] – can be used to derive the exact score numbers for the best results.

![Figure 2.1: Groups of amino acids by physical/chemical properties. Source: Alignment III – PAM Matrices from Introduction to Bioinformatics (COM S 544) course at Iowa State University 2014. www.cs.iastate.edu/~cs544/Lectures/PAM_matrices.ppt at Jan 18. 7:40 PM, 2015](image)

Although, in order to get more accurate alignments, we should take into account the fact that because of underlying biological and/or chemical reasons, not every pair of different amino acid pairs have the same probability of being transformed into one another. For example, it is more likely that neutral evolution would accept a mutation where a hydrophobic amino acid is replaced by another hydrophobic one than with a hydrophilic fragment. There are many different characteristics of amino acids that could be taken into account when we would like to assess their similarities; for a summary of some possible clusters
2. SEQUENCE ALIGNMENT METHODS

see Figure 2.1 (source: [8]).

PAM matrices

*Point Accepted Mutation* (PAM) means a simple substitution-type mutation within a protein sequence, which has been accepted by the natural selection processes of evolution. In other words, these are mutations that persisted: silent mutations of the DNA are not included, and neither are the lethal (or very detrimental) ones. Based on this concept, Margaret Dayhoff introduced a series of PAM matrices in 1978 [9].

Each row and each column of a PAM matrix represents a certain amino acid. Each entry in a 20 x 20 matrix corresponds to the likelihood that an amino acid is transformed to the other one (after some time had passed, allowing evolution to “work”), based on observed frequencies in accepted alignments of 71 families of proteins.

Before describing the method to construct PAM matrices, it is worth pointing out the benefits of choosing alignment scores as log-odds scores. The reasons are technical and practical: this way we can avoid multiplying many small numbers (exact probabilities) in a statistical computation. This means that we do not have to deal with the possible inaccuracies related to the limits of precision of floating point numbers. Also, by using log-odds, instead of multiplying we can add up the negative logarithms of the probabilities, which makes the computations faster. As Sean R. Eddy writes in [7]:

“What we want to know is whether two sequences are homologous (evolutionarily related) or not, so we want an alignment score that reflects that. Theory says that if you want to compare two hypotheses, a good score is a log-odds score: the logarithm of the ratio of the likelihoods of your two hypotheses. If we assume that each aligned residue pair is statistically independent of the others (biologically dubious, but mathematically convenient), the alignment score
is the sum of individual log odds scores for each aligned residue pair. Those individual scores make up a 20x20 score matrix”.

Building up PAM matrices

The basic idea is to construct a simple Markov process that will determine the probabilities of state transitions at each position of a sequence independently. To get the transition probabilities, we first need some statistics collected from aligned data (\(N\) is the total number of all amino acids in the database):

- total numbers of amino acid occurrences: \(n_j\)
- number of transformations (accepted point mutations) from amino acid \(j\) to \(i\) : \(A_{ij}\)
- frequencies of amino acid occurrences \(f_j = n_j/N\)
- mutability values of amino acids \(m_j = \sum_{i=1,i\neq j}^{20} A_{ij}/n_j\)

Now our goal is to construct a stochastic matrix \(M\) so that the \(M_{ij}\) values correspond to the probability of the \(j\)-th amino acid mutating into the \(i\)-th amino acid. \(\lambda\) is a constant scaling factor.

\[
M_{ij} = \lambda A_{ij} \frac{m_j}{\sum_{i=1,i\neq j}^{20} A_{ij}} = \frac{\lambda A_{ij}}{N f_j}
\]

The diagonal elements will contain the probability that the residue is unchanged:

\[
M_{jj} = 1 - \sum_{i=1,i\neq j}^{20} M_{ij} = 1 - \lambda m_j
\]

To standardize a unit of evolutionary distance, the PAM unit was being introduced by Dayhoff et al. One PAM unit is the amount of evolution that is expected to change 1% of the characters within a protein sequence. It was chosen to represent a small distance, so that the probability of superimposed changes (when more mutations occur at the same position) is negligible. The
indices of the PAM matrices within the series refer to PAM units (for example the PAM10 matrix will contain log-odds scores that are best to be used for protein alignment when the sequences are expected to have diverged to 10 PAM units: 10% of their amino acids differ).

The $\lambda$ proportionality factor corresponds to evolutionary time: a larger value implies more changes, thus a longer evolutionary distance. But how to find a good $\lambda$ value for a specific PAM matrix, for example for PAM1? In this case we want 99% of the amino acids to be conserved, so we sum up $n_j M_{jj}$ values (the numbers of conserved amino acids):

$$\sum_{j=1}^{20} n_j M_{jj} = \sum_{j=1}^{20} n_j - \lambda \sum_{j=1}^{20} n_j m_j = N - N\lambda \sum_{j=1}^{20} f_j m_j$$

So, the value of $\lambda$ for our first $M = M^{(1)}$ matrix (to be transformed to PAM1 soon) can be computed from:

$$0.99 = 1 - \lambda \sum_{j=1}^{20} f_j m_j$$

We still have probabilities in these $M^{(k)}$ matrices, and they are not necessarily symmetric. To finally compute the symmetric PAM$k$ matrices containing log-odds scores, we need to transform the values a bit further. We need the ratio of two probabilities: the probability of point accepted mutations substituting the $j$-th amino acid with the $i$-th amino acid, and the probability of these amino acids $i$ and $j$ being aligned together by pure chance; and then finally take a logarithm:

$$PAM_{kij} = \log \frac{f_j M^{(k)}_{ij}}{f_i f_j} = \log \frac{M^{(k)}_{ij}}{f_i}$$

For computing the subsequent matrices in the PAM series we can use that
2.1 Theory and practice of sequence alignment

Figure 2.2: PAM250 matrix from Dayhoff et al.’s publication [9]

we have a Markov process (meaning that we assume that the mutations are independent of each other, and the process has no memory, so the state in a next step depends on only the actual state and the state transition probability matrix), so we can write:

\[ M^{(k)} = M^{(1)}k \]

Usually PAM250 matrix is used if we have no previous knowledge of the expected similarity of the sequences or if they are likely to have substantially diverged (different at about 80% of the sites). If a closer relationship is suspected then PAM matrices with lower indices are used.
2. SEQUENCE ALIGNMENT METHODS

BLOSUM matrices

In the beginning of the 1990s, when much more protein data were available than in the late 70s, the authors of “Amino acid substitution matrices from protein blocks” [10], Henikoff and Henikoff thought that a better series of log-odds matrices could be crafted for scoring protein alignments. They took the BLOCKS database of multiple alignments, and counted substitution pairs within conserved regions of protein families, then followed similar steps of computation to what we have seen in the previous section.

The construction of BLOSUM (BLOcks SUbstitution Matrix) matrices is similar to that of the PAM series in the sense that these are also based on reliable alignment data and statistics. The key differences are:

- BLOSUM values are not extrapolated; PAM matrices are extrapolated from counts of amino acid mutations between very closely related proteins.

- BLOSUM mutation frequencies are counted based on local alignments (only specific, highly similar regions within the sequences); PAM values are based on global pairwise alignments.

- Matrices with higher index in the BLOSUM series correspond to higher similarity; larger indices within the PAM series mean a larger evolutionary distance, thus less similarity.

To get a series of scoring matrices, different sets of alignments with varying maximum difference were used to compute the frequency values. For example, BLOSUM62 was built based on aligned sequences with at most 62% similarity. This matrix proved to be very useful when aligning sequences about which we have no prior knowledge, so it became the default setting for many of the popular alignment tools. An interesting “fun-fact” is that the widely used BLOSUM62 is miscalculated – it is not accurate according to the published
2.1 Theory and practice of sequence alignment

Algorithm that described its creation; but this error was leading to an increase in alignment accuracy [11].

BLOSUM80 is used for more closely related sequences, and BLOSUM45 is applied in the case of more divergent proteins.

Table 2.1 summarizes the connection between the two scoring matrix sequences. Any pair of matrices occupying the same row are to be used in a similar context with regard to expected sequence similarity.

Custom substitution matrices

The series of BLOSUM and PAM matrices were crafted for general usage, based on data from alignments of a wide range of protein families. Therefore, if we have to deal with sequences that have biased amino acid compositions, then the standard substitution matrices are not optimal. Whenever the need for very accurate alignments of a specific subset of proteins arises, it might be worth

![Figure 2.3: BLOSUM62 matrix from Henikoff and Henikoff’s publication [10]](image.png)
2. SEQUENCE ALIGNMENT METHODS

<table>
<thead>
<tr>
<th>PAM</th>
<th>BLOSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAM100</td>
<td>BLOSUM90</td>
</tr>
<tr>
<td>PAM120</td>
<td>BLOSUM80</td>
</tr>
<tr>
<td>PAM160</td>
<td>BLOSUM60</td>
</tr>
<tr>
<td>PAM200</td>
<td>BLOSUM52</td>
</tr>
<tr>
<td>PAM250</td>
<td>BLOSUM45</td>
</tr>
</tbody>
</table>

Table 2.1: Comparing PAM and BLOSUM matrices

Considering building a targeted scoring matrix for the task.

Compositional score matrix adjustment, described in [12] by Altschul et al. does not start creating a new matrix from scratch, but instead makes smaller adjustments to one of the standard matrices, based on data and a probabilistic model. Compositionally adjusted matrices can not only give better alignments, but are useful also in the context of general protein database similarity searches.

Given enough data (three dimensional structures of at least some of the proteins from a family) it might be worth creating a specific substitution matrix for a separate group of proteins. This way, the tertiary foldings and secondary structures, thus the implied conserved domains are taken into account with a larger extent [13].

From this idea it is only one more step to take forward to use different matrices for aligning different domains of the proteins. For example, a domain-specific matrix has been developed for aligning transmembrane domains. Because of the hydrophobic nature of membrane spanning segments, this mutation data matrix is quite different from matrices calculated from general sequence sets which are biased towards water-soluble globular proteins [14].
2.1 Theory and practice of sequence alignment

2.1.2 Penalizing gaps

**Constant gap penalty**

The simplest method for penalizing gaps is to assign a fixed constant negative score (in the common case when we aim to find a high-scoring alignment with our algorithm; otherwise of course it would be a positive score) to each gap, regardless of the gap length.

**Linear gap penalty**

Linear gap penalty uses a negative score that grows linearly with the gap length: if a gap is \( l \) characters long, and the linear gap penalty value is \( P_{\text{lin}} \) then

\[
G(l) = lP_{\text{lin}}
\]

score is added to the alignment score because of this gap. The best value of \( P_{\text{lin}} \) to be assigned depends on the substitution matrix, and it is usually in the range of small negative integers (\(-1, -2, \text{maybe} -3\)). The drawback of this method is that long indels are given an unacceptably low score, so very often the result is a “gappy” alignment with many short indels, especially if the sequences are somewhat diverged – and this is usually not a reasonable alignment.

**Affine Gap Penalty**

This is the most frequently used gap scoring scheme. It combines the previous two: the penalty score for a gap of length \( l \) is computed as

\[
G(l) = P_{\text{op}} + lP_{\text{ext}}
\]

where \( P_{\text{op}} \) is the *gap opening penalty* and \( P_{\text{ext}} \) is the *gap extension penalty*.

This scoring method is still easy to implement, but offers a biologically more
2. SEQUENCE ALIGNMENT METHODS

exact model, it takes into account the fact that most indels are fairly short, but still, usually it makes no sense to have many tiny gaps very close to each other. Penalizing gap openings with scores 5–10 times more than extending an existing gap (usually $P_{\text{ext}} = -1$) makes it worth to join close small gaps together, and thus we get “nicer” alignments. The downside is, that if point deletions or insertions happen, they are over-penalized.

**Convex gap penalty**

Since the observed gap lengths follow a power law distribution, one might believe that the affine gap penalty might be improved by penalizing very long indels somewhat less:

$$G(l) = P_{\text{op}} + P_{\text{ext}} \ln(l)$$

This scheme requires somewhat more computation, and, unfortunately, in contrast to initial expectations, logarithmic gap costs often produce poorer alignments than the simpler affine method [15].

2.1.3 Homology versus similarity

Although the goal of sequence alignment is to discover evolutionary history of genes or proteins, it must be noted that homology (shared ancestry between a pair of structures, or genes, in different species) is not the same as molecular similarity. Usually there is a strong correlation between the two, but many examples exist where similar fuctions have evolved independently; and where – despite a quite close common ancestor – the fuctions of the proteins have diverged. As an example of the latter case: Zeta-crystallin (a component of the transparent lens matrix of the vertebrate eye) and E. coli’s quinone oxidoreductase metabolic enzyme are homologs, but despite the common, close ancestry, their function has changed during evolution [3].
2.1 Theory and practice of sequence alignment

On the other hand, we talk about *convergent evolution* when similar functional structures appear independently, without a common ancestor also having the structure. Three examples:

- Maybe the most widely known example is the camera-like eye of cephalopods (e.g., squids), vertebrates, and cnidaria (e.g., box jellies).
- Opposable thumbs have evolved at least twice independently: in primates, and in the – only distantly related – opossums.
- More mammal groups have independently evolved prickly protrusions of the skin (hedgehogs, and echidnas, for example).

What we can aim for, using alignment methods of sequence data (DNA or protein sequences), is only finding molecular similarity. The biological relevance of the *in silico* identified similarities usually must be evaluated by *in vitro* experiments.

### 2.1.4 Pairwise alignment

If we have two DNA or protein sequences, we might ask the questions: how closely related they are, or how long ago did their closest common ancestor live? We might be also interested in the series of mutational steps that could have led to these sequences.

A general approach to answer these questions is called *pairwise sequence alignment*, during which we try to find a best possible matching of characters within the sequences.

Although the number of possible alignments of two sequences grows exponentially with the length of the sequences, finding the best scoring alignment of two sequences is computationally feasible, since such an alignment can be found by iteratively comparing the prefixes of the two sequences [16]. The optimal
alignment of longer prefixes can be calculated quickly from shorter prefixes, and hence, the algorithm needs only memory and running time that both are proportional to the product of the lengths of the sequences.

Let us see a simple example of pairwise alignment:

| P53 HUMAN | MEEPSDBSVEPPLSQETFSDLKLPENNVLSPLPSQAM |
| P53 MARMO | MEEASDSLIEEPPLSHETFSDLWNNLPPENNVLSPVLSPPR |
| P53 HUMAN | DDLMLSPDDIELWFTEDPGDEAEPKMEAAAPAAPAPAAPAAP |
| P53 MARMO | DDLLSSEDVEENFDK---GPDEALQMSAAPKAPTPAAS |
| P53 HUMAN | TPAAPAPAPSWPLSS |
| P53 MARMO | TLAARSPATSWPPLSS |

It’s time to define a pairwise sequence alignment more formally. Let $X$ be a protein sequence of length $n$. The $i^{th}$ letter (or amino acid) of $X$ is denoted by $X_i$, and $X^j_i$ denotes the substring of $X$, from letter $i$ to $j$: $X_i X_{i+1} \ldots X_j$. (Note: if $i > j$ then $X^j_i$ is the empty string with length 0). Similarly, let $Y$ denote our second sequence, with length $m$.

When we align these two sequences, we get a series of aligned columns, consisting of two characters: the first is an amino acid from $X$ or a gap; the second is a character from $Y$ or a gap; but both can not be gaps at the same time; and the order of letters within $X$ and $Y$ can not be changed as we look at the columns in order.

To summarize from another point of view, the four types of columns are:

- a pair of matching letters ($X_i = Y_j$)
- a pair of nonmatching letters, a mismatch ($X_i \neq Y_j$)
- an insertion (a gap is introduced into $X$)
- a deletion (a gap is introduced into $Y$)

The length $l$ of the alignment will be at least the length of the longer sequence ($l \geq \max(m, n)$).

A score is assigned to a matched or mismatched pair of amino acids (scores are determined by a substitution matrix), and a penalty is given to a gap,
caused by an insertion or deletion in one of the sequences. The final alignment score is computed by summing all scores and gap penalties in the columns, and an optimal (highest scoring) alignment can be obtained efficiently using dynamic programming [17].

From other fields of computer science one might be familiar with algorithms computing edit distance between two strings. Here we have similar methods, the basic idea is the same, but the algorithms have been slightly modified to take into account the specialities of biological reality. The most obvious difference is that instead of a distance (or dissimilarity), here we usually compute a similarity score.

Global and Local Alignment

Approaches to pairwise sequence alignment generally fall into two broad categories:

- global alignments
- local alignments

Calculating a global alignment is a form of global optimization that attempts to align the entire length of all query sequences. By contrast, local alignments aim to identify shorter regions of high similarity within long sequences that are often widely divergent overall.

The question often rises: should we use local or global alignment for more accurate results? The answer depends on the relative length of similar parts of the sequences (their “cores”), and their relative positions. Not surprisingly, global methods usually perform better if the sequences share a similar “distribution” of their cores (that means, no conserved regions have been transposed for example), but local alignments give more accurate information about com-
mon ancestry when the “cores” were positioned at different places within the sequences [18].

A variety of computational algorithms have been applied to the sequence alignment problem. These include slow but formally correct methods like dynamic programming.

**Needleman-Wunsch algorithm**

The Needleman–Wunsch algorithm is one of the first applications of *dynamic programming (DP)* to align biological sequences. DP can be used in optimization problems where it is possible to quickly give an optimal solution to a complex problem by using already computed optimal solutions to smaller parts of the same problem. DP is usually applied to optimization problems where subproblems share subproblems. In contrast to simple recursive “divide and conquer” type of solutions, DP solves each subproblem exactly once, then stores the sub-results in a table from where they are easily looked up [19]. This way we can usually get a polynomial algorithm instead of the naive, exponential one.

The Needleman–Wunsch algorithm performs *global alignment*. Here is a quick example where we align two short DNA strings:

seq-1  Y E K P C A
seq-2  E K P C W A

To use our previous notations: X is seq-1 and seq-2 is denoted by Y; their lengths are \( n = m = 6 \). It is not hard to see that the optimal alignment (with any reasonable scoring matrix and gap penalty values) will be (also see Figure 2.4):

<table>
<thead>
<tr>
<th>seq-1</th>
<th>Y E K P C</th>
<th>- A</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq-2</td>
<td>- E K P C</td>
<td>W A</td>
</tr>
</tbody>
</table>
2.1 Theory and practice of sequence alignment

But how do we find this alignment automatically from the almost nine thousand possible ways? Guillaume Filion has explained the Needleman–Wunsch method with nice examples on his bioinformatics blog, *The Grand Locus*. In his words:

“The key of the algorithm is to represent an alignment by a path in a matrix, as illustrated by the figure [2.4]. The two sequences to align are written at the top and left margins of the matrix. Starting from the top left corner, a path is drawn to the bottom right corner. A horizontal move consumes one character from the first sequence [a deletion], a vertical move consumes one character from the second sequence [an insertion], and a diagonal move consumes one character from both. [...] We can define a simple score to measure the quality of an alignment as the number of matches minus the number of mismatches (including insertions and deletions). It then makes sense to speak of the ‘best’ alignment.”

Before building up the idea that allows us to fill in the DP table with partial substring-match-scores, let us note that there are three different ways to reach
the last, bottom right corner of the matrix and these represent the three ways to finish the alignment between two sequences (also see Figure 2.5):

![Possible last steps in a dynamic programming table: match, insertion or deletion.](http://blog.thegrandlocus.com/2014/06/once-upon-a-blast)

**Figure 2.5:** Possible last steps in a dynamic programming table: match, insertion or deletion.
Source of image: http://blog.thegrandlocus.com/2014/06/once-upon-a-blast at 03. 02. 2015 11.20.45 CET

**Ending with a match (or mismatch):** in this case the last letter of the first sequence is matched to the last character of the second sequence. To solve the whole alignment problem what is left to align the two remaining sequences: $X_1X_2 \ldots X_{n-1}$ and $Y_1Y_2 \ldots Y_{m-1}$

**Ending with an insertion:** the last letter of the second sequence is matched to a gap, because that last letter has been inserted (with regard to the first sequence). Now we have to solve a bit smaller problem of aligning $X_1X_2 \ldots X_n$ with $Y_1Y_2 \ldots Y_{m-1}$

**Ending with a deletion:** this is the mirror case of the previous one. What is left: to align $X_1X_2 \ldots X_{n-1}$ with $Y_1Y_2 \ldots Y_m$

Out of the three last steps we would like to choose the best (highest scoring) one. The three kinds of last steps can be repeated after “removing” a character from one or both sequences, thus we get a way to recursively count a score for the alignment. We also have to note for each position what step we have chosen of the three, so that the final solution would be easy to trace back from the bottom right position.

To initialize the algorithm (assuming linear gap penalty with $P_{lin} = -1$), we need to fill the first column and row with the values shown in Figure 2.6.
Then, we can fill the table going from left to right and top to bottom: at each \((X_i, Y_j)\) position we need to take the maximum of the three possible ways of getting there, by looking at the already filled positions of \((X_{i-1}, Y_{j-1})\), \((X_{i-1}, Y_j)\) and \((X_i, Y_{j-1})\), plus adding a score assigned to this last step: -1 if it is an insertion or deletion; and a match score (usually from a substitution matrix) if it is a match.

For brevity, I do not include here a more formal definition of the algorithm. In Chapter 3 I will describe a more complex dynamic programming algorithm in more detail.

**Smith–Waterman algorithm**

The Smith–Waterman algorithm is similar to the Needleman–Wunsch method, but it performs local sequence alignment. Instead of trying to align all characters from both sequences, it only aims to find highly similar regions within nucleotide or protein sequences. This approach is useful when we compare more distant sequences and we are looking for some common conserved domains.

The algorithm runs in a similar way, filling in a dynamic programming
table (Figure 2.7); but here we will have only non-negative scores. Wherever we would have written a negative value in the Needleman–Wunsch algorithm, now we write a zero. This way we can “start over” from any position, thus we have a chance to find local similarities, even if they are not located at a somewhat similar position within the original sequences.

After we have filled in the DP table, we look for local maxima within the matrix, and follow the paths backwards to extract the best local alignments.

2.1.5 Multiple sequence alignment

The multiple sequence alignment (MSA) problem is still considered as the Holy Grail of bioinformatics [20]. There are 547357 sequences in 2015’s first release
of the SwissProt\(^1\) protein sequence database, while on the other hand, there are only \(98243\) known structures in the last PDB\(^2\) database release. Therefore, there is still demand for *in silico* prediction of protein structures, and the protein structure prediction methods still need more accurate alignments [21].

A succinct definition of MSA from Edgar and Batzoglou’s 2006 paper [4]:

“A multiple sequence alignment (MSA) arranges protein sequences into a rectangular array with the goal that residues in a given column are homologous (derived from a single position in an ancestral sequence), superposable (in a rigid local structural alignment) or play a common functional role.”

Although these three criteria are essentially equivalent for closely related proteins, sequence, structure and function diverge over evolutionary time and different criteria may result in different alignments. Manually refined alignments continue to be superior to purely automated methods; there is therefore a continuous effort to improve the biological accuracy of MSA tools. Additionally, the high computational cost of most naive algorithms motivates improvements in speed and memory usage to accommodate the rapid increase in available sequence data.

\[
\begin{array}{c}
\text{sp|P0CH24} \\
\text{gi|ABC74991} \\
\text{sp|B2KPN7} \\
\text{tr|U6C2F8} \\
\text{tr|U6BZL2} \\
\text{sp|P0CH24} \\
\text{gi|ABC74991} \\
\text{sp|B2KPN7} \\
\text{tr|U6C2F8} \\
\text{tr|U6BZL2} \\
\end{array}
\]

Formally, a global alignment of \(K > 2\) sequences \(S_1, S_2 \ldots, S_K\) over an alphabet \(\Sigma\) is a \(K \times L\) matrix \(A\) that contains elements of the original alphabet

\(^1\)http://expasy.org/sprot/
\(^2\)http://www.pdb.org/pdb/home/home.do
2. SEQUENCE ALIGNMENT METHODS

and a “blank” character (usually denoted by a dash):

$$A = \{a_{ij}\}, a_{ij} \in \Sigma \cup \{-\}$$

such that ignoring the blank '-' characters, the $i^{th}$ row reproduces sequence $S_i$, and there exists no column consisting only of blanks in $A$. A maximal run of adjacent blank characters in a row is called a gap [22].

We could look at the multiple sequence alignment problem as a simple extension of the pairwise case. The problem with this approach is that the size of the dynamic programming table will grow exponentially with the length of sequences as we add more and more proteins or genes to the input set. This makes the problem computationally unfeasible even for a couple of tens of sequences.

The multiple sequence alignment can be divided into two distinct problems:

**the scoring problem:** how to score the alignments such that the best scored alignment is the most accurate one

**the algorithmic problem:** how to find efficiently the best scored (or at least a very close sub-optimal) alignment.

The scoring problem itself consists of three subproblems:

- scoring amino acid similarities (see 2.1.1)
- scoring gaps (see 2.1.2)
- specific rules for scoring multiple alignments (see 2.1.5).

Regarding the algorithmic aspect: many solutions are known to the MSA problem, that are in some way extensions of a pairwise alignment algorithm. Unfortunately, the naive dynamic programming-based algorithm becomes too slow too soon, since analyzing all possible combinations of prefixes requires
exponential \((O(L^K))\) where \(L\) is the length of sequences to be aligned, and \(K\) is the number of sequences) memory and running time.

The problem of multiple sequence alignment is NP-hard (proved for the case of Sum-of-Pairs scoring scheme in [5]), so if we need to quickly align a higher number of sequences, then we need to use heuristics, and accept approximate, sub-optimal solutions.

An aligned set of sequences often serves as an indispensable input for subsequent analysis – for example, finding highly conserved subregions or inferring evolutionary history – therefore it is essential to create as precise alignments as we can, within reasonable time and resource limits.

Scoring schemes

Whenever we mention the quality of an alignment, we should specify an exact scoring scheme under which we would like to evaluate the alignments.

Many different scoring schemes have been suggested so far: among them, the Sum-of-Pairs score, in short, SP-score or SP-measure seems to be the most sensible, and it has received a lot of attention [5].

The SP-score is – as its name suggests – a summation: it can be computed as the sum of the scores of pairwise global alignments induced by the MSA in question.

Following Reinert et al.’s definition [22], by \(A_{i_1,i_2,...,i_n}\) we denote the projection of \(A\) to the sequences \(S_{i_1}, S_{i_2} \ldots S_{i_n}\). The score of a pairwise projection of the MSA matrix \(A\) to a pair of sequences, \(S_k\) and \(S_{k'}\) is denoted by \(c(A_{k,k'})\) The pairwise score is usually defined as the sum over all match/mismatch weights of the character pairs standing in the same column (weights taken from a substitution score matrix), plus a penalty for each gap.

Using these notations, the final SP-score of a multiple sequence alignment
2. SEQUENCE ALIGNMENT METHODS

A can be defined as:

$$SP(A) = \sum_{k < k'} c(A_{k,k'})$$

Another way to look at computing the SP score is to sum the scores from all the columns, where a column’s score consists of the corresponding pairwise scores. In case of a linear gap penalty this essentially means swapping summations (let $l$ denote the length of the alignment):

$$SP = \sum_{j=1}^{l} \sum_{k < k'} score(a_{kj}, a_{k'j})$$

Of course, if we intend to use a more sophisticated gap scoring scheme, like the affine gap penalty system, then the score of a column cannot be computed without taking into account the previous column, because the penalty of a gap will differ if a new gap is started in this column.

Computing the optimal multiple alignment under SP-measure is proven to be NP-hard [5], but fortunately, fast heuristics exist [22].

**Heuristics: corner cutting**

Finding the best scoring multiple alignment under the *Sum-of-Pairs* scoring scheme is proven to be NP-hard [5], therefore it is very unlikely that any polynomial algorithm exists for the exact multiple sequence alignment problem.

The memory requirement and running time can be reduced by corner-cutting methods, that try to define a narrow strip in the multi-dimensional dynamic programming table which contains the optimal alignment. For more details on corner-cutting see 2.2.

**Heuristics: progressive alignment**

A simple but quite efficient solution to limit the exponential problem space of MSA is to use the *progressive* approach (also known as the *hierarchical* or
2.1 Theory and practice of sequence alignment

tree method). It works by building up the final alignment from consecutive pairwise alignments corresponding to a guide tree, that reflects the evolutionary relationships. I will explain the details in section 2.3.

Comparing alignments

When we aim to measure the accuracy of an alignment method, we need data that contains some alignments that are “golden standards”, so that we can compare the result alignment – that we got from the algorithm under study – to them. This process requires assigning a score to a pair of MSA-s, measuring their similarity, or distance.

A simple one of such methods is the True Column (TC) score, that counts the number of columns that appear without any change in both multiple alignments (without differentiating between the positions of the gaps). The problem with this is that even one mis-aligned sequence is able to ruin this score.

Two commonly used alignment accuracy measures are the developer score $f_D$ and the modeler score $f_M$, introduced in [23]. The first, $f_D$ measures the sensitivity (or “recall” in data mining terms): number of correctly matched character pairs divided by the number of matched pairs in the reference alignment. The second, $f_M$ computes specificity: it is the number of correctly matched pairs divided by the number of all matched pairs in the predicted alignment [24]. Unfortunately, neither of these two scores account for gaps. Other scores have been advised, for example, the “agreement score” in [25], that does take gaps into account, but it is not symmetric, therefore it is not a metric.

Ideally, we would like to use a metric: $d(A, B) \geq 0$; $d(A, B) = 0$ only if $A$ and $B$ are equal; it should be symmetric: $d(A, B) = d(B, A)$; and finally, the triangle inequality should hold: $d(A, C) \leq d(A, B) + d(B, C)$.

A measure holding up to these expectations is the AMA score, introduced by
2. SEQUENCE ALIGNMENT METHODS

Schwartz et al. [24]. It measures the proportion of correct pairwise homology statements, and its value lies in the fact that it possesses no inherent bias towards long or short alignments [26]. To define the AMA score metric, we first define the following sets corresponding to a multiple alignment $A$:

$$
\mathcal{H}_H(A) = \{ (c_j, c_k) | \exists i : (A_{ij} \neq -), (A_{ik} \neq -) \} \quad \text{pairwise homology statements}
$$

$$
\mathcal{H}_D(A) = \{ (c_j, 0) | \exists i : (A_{ij} \neq -), (A_{ik} = -) \} \quad \text{pairwise deletions}
$$

$$
\mathcal{H}_I(A) = \{ (0, c_k) | \exists i : (A_{ij} = -), (A_{ik} \neq -) \} \quad \text{pairwise insertions}
$$

$$
\mathcal{H}_N(A) = \mathcal{H}_D(A) \cup \mathcal{H}_I(A) \quad \text{pairwise non-homology statements}
$$

where $i$ stands for a column index, $c_j$ is a character index in the $j^{th}$ sequence, $c_j \in \{1, \cdots, |A_j|\}$, and $|A_j|$ is the length of the $j^{th}$ sequence. With these definitions, the accuracy of a predicted alignment, $P$, relative to the true alignment, $T$, is given by

$$
AMA(P, T) = AMA(T, P) = \frac{2|\mathcal{H}_H(P) \cap \mathcal{H}_H(T)| + |\mathcal{H}_N(P) \cap \mathcal{H}_N(T)|}{(K-1) \sum_k |A_k|} \quad (2.1)
$$

2.2 Corner cutting methods

Sequence alignment – especially in the case of aligning many, or long sequences – requires a lot of computational time. To reduce the computational time that would be necessary to reach an optimal solution, heuristics are used to speed things up. A subset of these are the corner cutting methods (sometimes also called “constrained sequence alignment”).

A dynamic programming algorithm builds up an alignment in a step-by-step manner by aligning longer and longer sub-sequences. This algorithm can be improved in terms of speed by discarding low-scoring partial alignments that
are very unlikely to be a part of a final optimal alignment. These can be found in the top right and bottom left corner of the pairwise dynamic programming table – this fact is the reason for the name of this kind of heuristic [27].

Most of these heuristics use a test value to decide where to cut: a pre-defined upper limit for the evolutionary distance between the two sequences. These methods work only if the pre-set limit is above the actual evolutionary distance.

These algorithms are suitable for database searches when we want to get similar sequences to a selected one: the test value can be an upper limit of the distance from the given sequence.

In the next two subsections I will describe two classical corner cutting methods for pairwise alignment: Spouge’s algorithm from 1989 and Gusfield’s from 1997. Then we will see what can be done in case of a multiple alignment.

### 2.2.1 Spouge’s algorithm

In the algorithm that Spouge published in 1989 [28], only those $d_{i,j}$ elements are computed within the $n \times m$ dynamic programming matrix for which $d_{i,j} + \frac{|(n-i)-(m-j)| \ast g}{g} \leq t$, where $t$ stands for the test value, and $g$ is a negative score for insertions and deletions (long indels are not allowed).

The main idea is to leverage the fact that any path from $d_{i,j}$ to $d_{n,m}$ will include at least $\frac{|(n-i)-(m-j)|}{g}$ gaps. So if $t$ is smaller than the distance between the sequences then the algorithm will produce an exact alignment.

This method can be extended to work with concave gap functions as well.

### 2.2.2 Gusfield’s algorithm

Gusfield discusses a slightly different question in his book [20]. The $k$-difference global alignment problem can be defined as:
2. SEQUENCE ALIGNMENT METHODS

Given strings $S_1$ and $S_2$ and a fixed number $k$, the $k$-difference global alignment problem is to find the best global alignment of $S_1$ and $S_2$ containing at most $k$ mismatches and spaces (a space is a one-character long indel).

The basic approach is to compute the edit distance of $S_1$ and $S_2$ using dynamic programming but instead of filling up the whole $n \times m$ table, fill only an $O(km)$-size portion of the table.

The key observation is the following: if we define the main diagonal of the dynamic programming table as the cells $(i, i)$ for $i \leq n \leq m$, then any path in the table that defines a $k$-difference global alignment must not contain any cell $(i, i + l)$ or $(i, i - l)$ where $l$ is greater than $k$. (Note that this implies that $m - n \leq k$ is a necessary condition for there to be any solution.) Therefore, to find any $k$-difference global alignment, it suffices to fill in the table in a strip consisting of $2k + 1$ cells in each row, centered on the main diagonal.

If there is no global alignment of $S_1$ and $S_2$ with $k$ or fewer differences then the value obtained for cell $(n, m)$ will be greater than $k$, and it indicates that the computed edit distance is not necessarily the correct distance of $S_1$ and $S_2$.

2.2.3 Corner cutting in MSA

The corner cutting idea has been extended quite early to the case of multiple alignment [29, 30], with which the optimal alignment of 4-10, each 200–300 long sequences can be found in reasonable time [31].

As an example, let us follow the method described in [32], for the case when the Sum-of-Pairs scoring scheme is applied. If we denote the distance function over the set of sequence characters plus the gap character (−) by $d$ then the $k$-th aligned column’s score is

$$SP_k = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} d(a_{k,i}, a_{k,j})$$
2.3 Progressive alignment

where $n$ is the number of aligned sequences.

The $k$-suffix of a sequence means a subsequence from position $k$, to the end of the sequence. Let us denote the distance between the $k$-suffix of $i$-th sequence and the $l$-suffix of the $j$-th sequence by $w_{i,j}(k,l)$. Carillo and Lipman’s algorithm computes only positions where

$$d_{i_1,i_2...i_n} + \sum_{j=1}^{n} \sum_{k=j}^{n} w_{j,k}(i_j, i_k) \leq t$$

where $t$ is the test value.

The correctness of the algorithm can be seen by noting that the optimal SP-score of the not-yet-aligned suffixes of the sequences can not be higher than the sum of the pairwise alignment scores.

The $w_{i,j}(k,l)$ values can be computed from pairwise alignments, so with this heuristic the algorithm is able to compute the optimal alignment of six sequences of about 200 characters [33].

2.3 Progressive alignment

The vast majority of heuristics-based MSA programs align sequences using the progressive approach, combining global and/or local methods [34, 35, 36, 37]. This type of algorithm builds a MSA through a series of consecutive pairwise alignments, following the branching order of a guide tree (in bottom-up direction, from leaves to root), joining in more and more distantly related sequences. The progressive method has the drawback that once errors are introduced at an early step, they cannot be removed later [38]. This is especially conspicuous in the case of gaps: from the “once a gap, always a gap” rule (explained, for example, in [39]) it follows that gaps inserted into an alignment at an earlier step cannot be removed or modified further up in the guide tree,
2. SEQUENCE ALIGNMENT METHODS

often resulting in too many gaps, and a suboptimal alignment at the root of the tree [21].

The guide tree is usually constructed from the pairwise distances of the sequences that are computed using pairwise sequence alignments.

Note that given all the pairwise alignments between sequences, it is almost always impossible to create an MSA that is consistent with each and every pairwise alignment. For example, consider aligning these three very short sequences:

seq1 \(\text{A\ C\ G}\)
seq2 \(\text{C\ G\ A}\)
seq1 \(\text{G\ A\ C}\)

Their optimal pairwise alignments and the MSAs derived thereof are depicted on Fig. 2.8

<table>
<thead>
<tr>
<th>Merge using alignments with 1st sequence</th>
<th>Merge using alignments with 3rd sequence</th>
<th>Merge using alignments with 2nd sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>- a c g -</td>
<td>- a c g</td>
<td>a c G -</td>
</tr>
<tr>
<td>- - c g a</td>
<td>- g a c</td>
<td>- c G a</td>
</tr>
<tr>
<td>g a c -</td>
<td>c g a -</td>
<td>- G a c</td>
</tr>
</tbody>
</table>

Now the MSA is not consistent with the optimum pairwise alignments of the 2nd and 3rd sequences
Now the MSA is not consistent with the optimum pairwise alignments of the 1st and 2nd sequences
Now the MSA is not consistent with the optimum pairwise alignments of the 1st and 3rd sequences

*Figure 2.8*: Example where three different MSAs are produced from three pairwise alignments, but none of them is consistent with all of the pairwise cases.
2.3 Progressive alignment

The usual basic procedure consists of three steps\cite{40}:

1. Compute all pairwise alignment scores between the sequence pairs

2. Build a guide tree (for example, using the UPGMA\cite{41} or Neighbor-joining\cite{42} method)

3. Align the sequences sequentially, guided by the phylogenetic relationships (indicated by the tree)

Note that corresponding to the inner nodes above the second level, we need to align not sequences but alignments together. We can still perform this via a dynamic programming algorithm, but the computation and the scoring becomes more complex. I will present our method in Chapter 3.

Despite the inherent drawbacks – especially noteworthy in the case of aligning distantly related sequences –, the efficiency of the progressive method guarantees its wide adoption. Tools that use this heuristic (ClustalW, for example) allow us to align up to thousands of protein sequences within reasonable time.

2.3.1 Iterative alignment

Iterative methods build on top of the progressive approach: they try to avoid the traps of the basic method by re-aligning some parts of an alignment iteratively, until an upper limit of cycles has been reached, or until there is no improvement according to a pre-defined objective function \cite{40}. Sometimes these methods also update the guide tree during the iterative steps, based on the current best alignment, then use the new tree in the next round to compute a better alignment.

Among others, MUSCLE and MAFFT uses iterative techniques to improve the accuracy of their output alignments. Many methods are summarized and compared in \cite{43}.
2. SEQUENCE ALIGNMENT METHODS

2.4 Software tools for MSA

In the following subsections I will briefly introduce three widely adopted multiple sequence alignment tools. Their detailed comparison is not an easy task [44] and it would stretch far beyond the scope of this dissertation, so for the curious I suggest a recent paper by Pais et al. [38].

I tried all three options, and found that they are optimal for different use cases. At the end of each following subsection, I noted where and how I used these programs in the projects constituting this thesis.

2.4.1 Clustal

The Clustal series of software, developed by Higgins et al. since 1988 is one of the most widely used tools for multiple sequence alignment. Its original release, introduced in [36] needed a guide tree as an input for progressive alignment. The subsequent incarnations (starting from Clustal V [45], a complete rewrite in C language) were able to construct a phylogenetic tree from the input data, after performing pairwise alignments as a first step. The ClustalW (command-line) and Clustal X (graphical) versions incorporated a novel position-specific scoring scheme and a weighting scheme for down weighting over-represented sequence groups [46]. Clustal W version 2.0 included new options to allow faster alignment of very large data sets and to increase alignment accuracy [47].

The latest release (2011), Clustal Omega, is a command line only program, described in [48, 49] that is capable of aligning thousands of sequences faster and more accurately than other solutions. The project’s webpage (http://www.clustal.org/omega/) says that the program allows “hundreds of thousands of sequences to be aligned in only a few hours.” Usually the default settings give good results, but users have the option to customize the parameters (most commonly the gap opening penalty, and the gap extension penalty).
2.4 Software tools for MSA

An example output can be seen in Figure 2.9. Here, an ‘*’ (asterisk) indicates a position which has a single, fully conserved residue. A ‘:’ (colon) indicates conservation between groups of strongly similar amino acids – where the corresponding score exceeds 0.5 in the Gonnet PAM 250 matrix. A ‘.’ (period) indicates conservation between groups of weakly similar protein building blocks – where the PAM250 score is smaller than or equals to 0.5.

I will use Clustal Omega for fast alignment of large sets of protein fragments in Chapter 5.

![Figure 2.9: Clustal Omega colored alignment output](image)

2.4.2 MAFFT

MAFFT has been one of the most important challengers of Clustal since 2002. It promised drastically faster running time and not much worse accuracy, compared to Clustal [50]. MAFFT employs Fast Fourier Transform (FFT) to quickly identify homologous regions within the sequences, and implements two different heuristics: the progressive method (FFT-NS-2) and the iterative refinement method (FFT-NS-i).

In 2005, in version 5 new iterative refinement options were added, which took into account the pairwise alignments in the objective function, and with these, MAFFT became more accurate than Clustal on benchmark sets (while still being faster) [51].
2. SEQUENCE ALIGNMENT METHODS

Figure 2.10: Limitations of MAFFT: it becomes slow if the sequences are too distant or extended swaps had occurred between them

The most recent version has several new features, including options for adding unaligned sequences into an existing alignment, constrained alignment and parallel processing. The related paper [52] also clarifies MAFFT’s limitations (see Figure 2.10).

In a first-round implementation I tried to use MAFFT to generate alignments for evaluating our alignment DAG building method, described in Chapter 4. The problem with MAFFT was that it only gave us an alignment as output, but not the resulting MSA’s alignment score. We needed these scores to be able to guide the MCMC procedure, so we switched to MUSCLE instead.

2.4.3 MUSCLE

MUSCLE, developed in 2004, also uses iterative refinement after carrying out a progressive MSA. Elements of the algorithm include fast distance estimation using $k$-mer counting, progressive alignment using a new profile function (log-expectation score), and refinement using tree-dependent restricted partitioning.

Figure 2.11 shows an overview of the flow of the algorithm. Edgar’s paper [53] has been cited over 12000 times, being one of the best-performing multiple alignment programs according to published benchmark tests.

I used MUSCLE to generate alignments for measuring the accuracy and speed of our alignment DAG building method, described in Chapter 4.
2.5 Handling alignment uncertainty

Sequence alignment is often an important first step in a wide range of different analyses, including identification of conserved motifs [54], analysis of molecular co-evolution [55, 56], estimation of phylogenies [57], and homology-based protein structure prediction [58, 59]. Therefore, it would be desirable to have tools to estimate how much we can trust an alignment.

The most frequently used alignment methods seek to compute a single optimal alignment, using custom-tailored dynamic programming algorithms [16, 60] as well as a variety of heuristic procedures [53, 61, 62, 63]. Similar approaches can be used to find maximum likelihood alignments under certain probabilistic models of insertion, deletion and substitution events [64, 65, 66, 67, 68].

The standard approach of using a single multiple sequence alignment discards a great deal of important information contained in some sub-optimal
alignment parts regarding the evolutionary relationships between the sequences [69, 70].

Sensitivity to the alignment is also observed in many types of downstream analysis, for example:

- homology modeling of protein structures [71]
- detection of correlated evolution [72, 73]
- prediction of RNA secondary structure [74]
- inference of positive selection [75, 76, 77]
- phylogenetic inference [78, 79, 80, 81].

A common approach to tackling the issue of alignment uncertainty has been to discard unreliable regions of the alignment, in order to remove these before carrying out subsequent analysis. These filtering methods have been observed to yield improved inference for phylogenies [82, 83, 84] and positive selection [76, 77]. However, the choice of filtering method may also have a strong influence on the results [85], and uncertain regions of the alignment may contain important information that is lost.

Many different approaches have been developed in recent years to generate more alignments in parallel according to their probability, yielding information about the distribution of alignments rather than simply reporting a single optimum. (For some details and examples, see [26].) Despite these efforts, currently this type of probabilistic information is not widely used in the context of downstream inference. The most important reason for this might be the lack of tools and standardized methodology for making use of a distribution over alignments [26].

To incorporate the inherent uncertainties (an optimal alignment’s dependence on parameters, like the substitution matrix and indel scoring) into the
result of an alignment, we chose to develop a framework (both a theoretical one and an implemented software framework) for representing the space of sampled alignments. The set of selected output alignments are represented as a directed acyclic graph (DAG); each path through this graph then represents a valid multiple alignment, weighted by its probability. Since the probabilities of individual columns can be estimated more reliably than that of whole alignments, this approach allows for more accurate estimation of posterior probabilities.

An additional benefit of using DAGs is that the number of alignments that can be taken into account this way is much more than the number of sampled alignments. The sample size is effectively multiplied many-fold, because as the individual sampled paths are joined together at common alignment columns (as intersections), many more paths appear and become viable through the graph.

Details are presented in Chapter 4, and in Herman et al.’s paper: Efficient representation of uncertainty in multiple sequence alignments using directed acyclic graphs [26].

2.6 Introduction to protein search methods

With the advent of larger protein and genomic databases came the need for quick search within these databases by sequence similarity. A biologist may often start his or her investigation regarding a newly sequenced protein by looking up similar proteins from other species, for example.

In the next subsections I will briefly introduce two software packages addressing this problem, in their historical order.

2.6.1 FASTA

Lipman et al. were the first to give an efficient solution to mass sequence similarity calculation in [86]. The main idea of FASTA (originally FASTP) is
not to try to extensively align a query sequence to all other sequences from the
database – so many dynamic programming calculations would take a very long
time –, but to pre-filter possibly good candidates by a simpler pattern matching
approach.

The main idea is to match short words of fixed length exactly, and if many
these simple word matches are found within certain regions of the sequence
pair then it makes sense to perform a more sophisticated alignment. The short
matches are called seeds, which gives the name of this family of heuristics:
seed-based methods.

The problem with the FASTA algorithm is that it misses certain matches:
when the exact words that can match are never long enough, although overall
a high score could be achieved with a proper dynamic programming method.
An example depicted by Figure 2.12 (taken from Guillaume Filion’s blog [87])
shows a case of high similarity that can not be found if the word size is 4 or
above.

\[
\text{NEV-RMATCHI-GGREATE-THANTHR-E} \\
\text{NEVERMA-CHINGGR-ATERTHA-THREE}
\]

\textbf{Figure 2.12: Example}

\subsection*{2.6.2 BLAST}

The BLAST algorithm, developed by Altschul \textit{et al.} was developed for protein
and DNA/RNA sequence similarity searches [88]. The goal was to perform the
searches faster than FASTA, but not to lose in regards of sensitivity. BLAST
follows a heuristic as well, that almost always works to find related sequences
in a database search, but does not have the underlying guarantee of an optimal
solution like a dynamic programming algorithm. FASTA finds short common
patterns in the query and in database sequences and joins these into an alignment. BLAST is similar to FASTA, but gains a further increase in speed by searching only for rarer, more significant patterns in nucleic acid and protein sequences [40].

BLAST is very popular due to its availability on the World Wide Web through a public server at the National Center for Biotechnology Information (NCBI) and at many other sites. The BLAST algorithm has evolved to provide molecular biologists with a set of very powerful search tools that are freely available to run on many computer platforms [89].
2. SEQUENCE ALIGNMENT METHODS
Reticular Alignment

Note: This chapter is based on excerpts from the publication “Reticular alignment: A progressive corner-cutting method for multiple sequence alignment”. I re-used the images and some paragraphs (definitions, etc.) of the original text between quotation marks, but restructured the paper for brevity. Most of the Theorems and Proofs are removed, for these, see the full publication [21]. I will clarify in the last section which parts of this collaboration were my work.

3.1 Introduction

Motivated by the ever-rising need for more exact modeling tools in bioinformatics following the expanding limits posed by hardware and software technology, we proposed a new method that is capable of finding a nearly-optimal solution to the NP-hard problem of multiple alignment.

It is a variant of corner-cutting methods, combined with the idea of progressive sequence alignment guided by a phylogenetic tree. The novelty of our algorithm is that we do not restrict the search for solutions to a compact part of the dynamic programming table, but instead, as written in [21], “our ap-
3. RETICULAR ALIGNMENT

The algorithm was implemented in Java programming language to ensure platform independency. The quality of the final alignments was tested on the BAliBASE database [90], using a couple of scoring schemes. We compared our method with the most popular alignment tools:

- ClustalW [91],

![Figure 3.1: A small example of a reticular alignment network: it shows three different alignments of the sequences ALLGVGQ and AVGQ](image)
3.1 Introduction

- MAFFT [50]

- Fast Statistical Alignment (FSA) [92]

We found that Reticular Alignment could outperform ClustalW even if a simple scoring scheme was applied. With the use of sophisticated scoring models (decreasing gap penalties for hydrophilic parts, sequence weighting in sum-of-pairs scoring, etc.) Reticular Alignment outperformed FSA and even MAFFT in some accuracy measurements (see Figure 3.2).

![Figure 3.2](https://example.com/figure32.png)

**Figure 3.2:** Comparison of alignment software on BAliBASE v2.01 Refs 1-5. Alignment accuracy of multiple alignment programs compared to that of RetAlign as measured on BAliBASE v2.01 Reference sets 1-5 using the provided bali_score tool (SP and TC scores calculated on all of the columns versus on columns containing features are all shown). RetAlign was run with sequence weighting on, a single guide tree iteration and with a reticular threshold of 800. FSA was run in maximum sensitivity mode. MAFFT was run with the -auto switch and ClustalW with the default settings.
3. RETICULAR ALIGNMENT

3.2 Methods

In this section, I briefly introduce the concepts and algorithms which form the theoretical background of the Reticular Alignment tool.

3.2.1 The extended Waterman–Byers algorithm

The Waterman–Byers algorithm [93] is a corner-cutting procedure that returns all alignments that’s score fall below the score of the optimal alignment at most by a pre-defined constant value.

Before introducing our algorithm (a modified Waterman-Byers), in the following couple of paragraphs I introduce the basic notations and define the necessary concepts. Some will be a bit longer excerpt from the RetAlign paper [21], but I think these are necessary for understanding the key findings, and a rewording of the exact definitions would possibly hinder the understanding.

“Let $A$ and $B$ be two sequences over an alphabet $\Sigma$, of lengths $n$ and $m$, respectively. Let $A_i$ denote the $i$ long prefix of sequence $A$, and let $A'$ denote the suffix of $A$ starting in the $i+1^{st}$ position. In this way, $A_i \circ A' = A$, where $\circ$ denotes concatenation. Let $a_i$ denote the character of $A$ in position $i$.

Let $s : \Sigma \times \Sigma \rightarrow R$ be a similarity function. $g_o$ will denote the gap opening and $g_e$ will denote the gap extension penalty. The score of any alignment, and thus all introduced concepts based on the alignment scores depend on the choices of similarity function, gap opening and gap extension penalty. However, for sake of simplicity, we omit to denote this dependence.”

The generalised algorithm

Our modified Waterman–Byers algorithm is built up of three main parts:

1. Forward-align algorithm
2. Backward-align algorithm

3. Alignment search algorithm that finds all alignments that have at least a given score (it uses the scores calculated by the forward and backward algorithms)

The forward-align part calculates the score of the best possible alignment of prefixes $A_i$ and $B_j$ within the following three subsets of alignments:

- Alignments that end in two aligned (matched) characters. The score of the best alignment of the prefixes $A_i$ and $B_j$ in this set is $M_f(i, j)$.

- Alignments ending in a character insertion $b_j$. The score of the best alignment of prefixes $A_i$ and $B_j$ in this set is denoted by $I_f(i, j)$.

- Alignments that end with the deletion of character $a_i$. The best score achievable by aligning prefixes $A_i$ and $B_j$ in this way is denoted by $D_f(i, j)$.

Then the final score of the optimal alignment of prefixes $A_i$ and $B_j$ is

$$\max\{M_f(i, j), I_f(i, j), D_f(i, j)\}$$

Note that each of $M_f$, $I_f$ and $D_f$ can be calculated efficiently using a dynamic programming algorithm. The details are published in [21].

It is also important to mention that by using the forward and backward scores, it is possible to find all alignment columns that are a part of an alignment that has a score above a given threshold [21].

$x$-networks

This is how the $x$-network of the alignments is defined in [21]:
Definition 1. For any sequences $A$ and $B$, $x \geq 0$, the $x$-network of the alignments of $A$ and $B$ is a directed graph $G(V, E)$. The vertex set consists of alignment columns $\alpha$ for which $b(\alpha) \geq \text{opt} - x$, where $\text{opt}$ is the score of the optimal alignment of $A$ and $B$; plus two auxiliary vertices, representing the beginning and the end of the alignment. These two auxiliary vertices are denoted by Start and End. An edge is going from vertex $\alpha_1$ to vertex $\alpha_2$ if there is an alignment in which $\alpha_1$ is followed by $\alpha_2$. The outgoing edges from the Start vertex go to the alignment columns with which the alignment might start, and the incoming edges of the End vertex come from the alignment columns that might be at the end of an alignment.

It can be proven that an $x$-network never contains dead ends (for the proof, see [21]).

“An $x$-network can be constructed using an algorithm that first runs the forward and backward algorithm to calculate $b(\alpha)$ for each possible alignment column $\alpha$, selects those columns for which $b(\alpha) \geq \text{opt} - x$, and builds the network from them.”

3.2.2 Aligning a network of alignments to another network of alignments

We are going to extend the original Waterman–Byers algorithm to align a network of alignments to another network of alignments. The definitions needed before describing the modified algorithms are taken from [21].

At first we define the network of alignments:

Definition 2. A network of alignments of sequences $A_1, A_2, \ldots, A_k$, $k \geq 1$ is a directed acyclic graph whose vertices are alignment columns of the set of sequences together with a unique source (denoted by Start) and a unique sink
3.2 Methods

(denoted by End). The vertices along any path from the source to the sink form a multiple sequence alignment of the set of sequences.

Of course, any single sequence \((k = 1)\) can also be considered as a simple network. An \(x\)-network is also a network of alignments.

We can generalise the definition of the \(x\)-network of two sequences to the \(x\)-network of two networks of alignments. Before this, we first have to define the alignment of alignments.

**Definition 3.** An alignment of two alignments \(\mathcal{A}\) and \(\mathcal{B}\) of sequences \(A_1, A_2, \ldots A_k\) and \(B_1, B_2, \ldots B_l\) is a multiple sequence alignment of these \(k+l\) sequences such that the non all-gap columns of the first \(k\) rows gives back \(A\), and the non all-gap columns of the last \(l\) rows gives back \(B\).

During aligning a network to another using a dynamic programming algorithm, it is important to visit the alignment columns of the network in a pre-defined order. Otherwise, it would not be guaranteed that the necessary entries are already calculated by the time we want to use them in the dynamic programming recursion. Therefore, a linear extension of networks is needed, that can be used for traversing the network.

**Definition 4.** A linear extension of a directed acyclic graph is a total ordering, \(<\), on the vertices such that for any two vertices \(v\) and \(u\), if there is a directed path from \(v\) to \(u\) then \(v < u\).

Note that the forward-align and the backward-align algorithms work with prefix-alignments and suffix-alignments defined as:

**Definition 5.** A prefix-alignment is a prefix of an alignment achievable by aligning an alignment \(\mathcal{A} \in \mathcal{A}\) to an alignment \(\mathcal{B} \in \mathcal{B}\). Similarly, a suffix-alignment is a suffix of an alignment achievable by aligning an alignment \(\mathcal{A} \in \mathcal{A}\) to an alignment \(\mathcal{B} \in \mathcal{B}\).
3. RETICULAR ALIGNMENT

The following couple of paragraphs make use of the previous definitions, and describe the generalised Waterman–Byers algorithm. They are taken from [21] without modification.

“The input consists of a threshold value \( x \geq 0 \) and a couple of networks of alignments, \( A \) and \( B \), together with a linear extension for each network. The output is the \( x \)-network of \( A \) and \( B \) together with a linear extension of it.

The algorithm uses a forward and a backward dynamic programming algorithm. The forward align algorithm calculates the score of the best prefix-alignment in which the last non all-gap columns in the first \( k \) lines is \( a_i \) and in the last \( l \) lines is \( b_j \) for each subset of alignments:

- alignments ending with \( a_i/b_j \). The score will be denoted by \( M_f(i, j) \).
- alignments ending with \( -i/b_j \). The score will be denoted by \( I_f(i, j) \).
- alignments ending with \( a_i/-j \). The score will be denoted by \( D_f(i, j) \).

The initialisation is:

\[
M_f(0, 0) = 0 \quad (3.1)
\]
\[
I_f(0, 0) = D_f(0, 0) = -\infty \quad (3.2)
\]

The dynamic programming algorithm visits the vertices of the two networks in their linear order. The recursions are:
3.2 Methods

\[ M_f(i, j) = \max_{i' \in \mathcal{N}^*(i)} \max_{j' \in \mathcal{N}^*(j)} \{M_f(i', j'), I_f(i', j'), D_f(i', j')\} + s(a_i, b_j) \] (3.3)

\[ I_f(i, j) = \max_{j' \in \mathcal{N}^*(j)} \{M_f(i, j') + g(a_i, -i, b_j, b_j), I_f(i, j') + g(−i, −i, b_j, b_j), D_f(i, j') + g(a_i, −i, −j, b_j)\} \] (3.4)

\[ D_f(i, j) = \max_{i' \in \mathcal{N}^*(i)} \{M_f(i', j) + g(a_i', a_i, b_j, −j), I_f(i', j) + g(−a_i', a_i, b_j, −j), D_f(i', j) + g(a_i', a_i, −j, −j)\} \] (3.5)

where \( \mathcal{N}^*(i) \) is the set of indices of vertices sending an edge to the vertex indexed by \( i \), and \( g(a, b, c, d) \) is the gap penalty function for alignment column \( b/d \) preceded by alignment column \( a/c \). We assume that the gap penalty for a given alignment column can be calculated from the alignment column in question and its preceding alignment column. See details in the subsection Gap penalties below. The maximum of an empty set is defined to be \(-\infty\).

The backward algorithm calculates the following scores:

- \( M_b(i, j) \) denotes the score of the best suffix-alignment that can follow the alignment column \( a_i/b_j \). Furthermore, the gap score of the first alignment column is calculated as if \( a_i/b_j \) was inserted before the first alignment column.

- \( I_b(i, j) \) denotes the score of the best suffix-alignment that can follow the alignment column \( −i/b_j \). Furthermore, the gap score of the first alignment column is calculated as if \( −i/b_j \) was inserted before the first alignment column.

- \( D_b(i, j) \) denotes the score of the best suffix-alignment that can follow the alignment column \( a_i/−j \). Furthermore, the gap score of the first alignment column is calculated as if \( a_i/−j \) was inserted before the first alignment column.
column is calculated as if \( a_i/j \) was inserted before the first alignment column.

The initialisation of the dynamic programming algorithm is

\[
M_b(n, m) = I_b(n, m) = D_b(n, m) = 0 \quad \forall n \in N^+(End_A), m \in N^+(End_B) \quad (3.6)
\]

where \( End_A \) and \( End_B \) are the sink vertex of networks \( A \) and \( B \), respectively.

The dynamic programming algorithm visits the vertices of the two networks backward in their linear extension. The recursions are

\[
M_b(i, j) = \max_{i' \in N^-(i)} \max_{j' \in N^-(j)} \{ M_b(i', j') + s(a_{i'}, b_{j'}), \quad (3.7) \\
I_b(i, j) + g(a_i, -i, b_j, b_{j'}), \\
D_b(i', j) + g(a_{i'}, a_i, b_j, -j) \}
\]

\[
I_b(i, j) = \max_{i' \in N^-(i)} \max_{j' \in N^-(j)} \{ M_b(i', j') + s(a_{i'}, b_{j'}), \quad (3.8) \\
I_b(i, j') + g(-i, -i, b_j, b_{j'}), \\
D_b(i', j) + g(-i, a_{i'}, b_j, -j) \}
\]

\[
D_b(i, j) = \max_{i' \in N^-(i)} \max_{j' \in N^-(j)} \{ M_b(i', j') + s(a_{i'}, b_{j'}), \quad (3.9) \\
I_b(i, j') + g(a_i, -i, -j, b_{j'}), \\
D_b(i', j) + g(a_i, a_{i'}, -j, -j) \}
\]

where \( N^-(i) \) is the set of indices of vertices to which an edge is going from the vertex with index \( i \).”

In the RetAlign paper [21] we have proven that the best score of alignments containing \( a_i/b_j, -i/b_j \) and \( a_i/-j \) is \( M_f(i, j) + M_b(i, j), I_f(i, j) + I_b(i, j) \) and \( D_f(i, j) + D_b(i, j) \), respectively. Therefore, the maximal score, \( opt \), can be
calculated from the following equation:

\[
\text{opt} = \max_{n \in N^+ (\text{End}_A)} \max_{m \in N^+ (\text{End}_B)} \{ M_f(n, m), I_f(n, m), D_f(n, m) \}
\]  
(3.10)

The implementation of the RetAlign algorithm is based on these definitions and equations.

### 3.2.3 The Reticular Alignment algorithm

The Reticular Alignment algorithm consists of these four steps:

1. Loading or building guide tree (representing the evolutionary connections between the sequences).

2. Transforming the input sequences (at the leaves of the guide tree) into simple, linear ‘networks’.

3. Visiting the inner nodes of the guide tree in bottom-up order, and for each node \( v \) with children \( u_1 \) and \( u_2 \), calculating the \( x_v \)-network of \( A_1 \) and \( A_2 \) (the children’s networks of alignments) using the generalised Waterman–Byers algorithm.

4. Returning the best (highest scored) alignment from the \( x \)-network calculated at the root of the guide tree.

If we set \( x = 0 \), then only the locally optimal multiple alignments are propagated upwards between the tree nodes. In this case, the Reticular Alignment algorithm is essentially no different from a usual progressive multiple alignment method. When \( x \) is set to \( \infty \), the Reticular Alignment method performs an exhaustive search in the space of multiple alignments, thus it finds the globally
3. RETICULAR ALIGNMENT

optimal alignment (in case of sufficient memory and time allowed). As we increase $x$ from 0 upwards, the size of the network also increases, as well as the running time and memory usage.

Other parameters to be set by the user are (also see Figure 3.8):

- guide tree construction method (see subsection "The importance of a good guide tree" in [21])
- substitution matrix (similarity scoring of alignment columns)
- gap scoring scheme (not detailed here, see sections "Gap penalties" and "Comparing single and pairwise gap penalties" in [21])

Scoring similarities

We used the BLOSUM62 matrix [10] to score pairwise similarities between aligned amino acids, and used the sum-of-pairs (SP) scoring scheme. A similarity score for an alignment column is the sum of similarity values for each pair of occurring non-gap characters in the column. The final score of an alignment is computed as the sum of the similarity scores for all columns.

Note that this pairwise scoring method is slightly different from the approach of ClustalW. For details, see [21].

Selecting the threshold value

The accuracy of the alignments created by the Reticular Alignment method depends on the strategy for choosing an $x$ value for each alignment step at the internal nodes. For ergonomical reasons we chose to set the $x$ value dynamically such that the final size of the alignment network depends on the length of one of the best scored multiple alignments as well as on a user-defined parameter, $t$. This $t$ is the "Threshold to be passed for computation" value on the GUI.
of our application; the slider underneath is just for convenience, with a log transformation (see Figure 3.8).

The connection between the threshold $t$ and $x$ is that $x$ will be a value such that the size of the $x$-network at that particular tree node will be at most $(t + 100)$% of the length of a reasonable alignment (measured in number of alignment columns).

For any $t$ value, the corresponding $x$ value can be determined by building up the network gradually. Alignment columns to be considered are placed in a priority queue, sorted by the score of the best alignment they appear in; then groups of columns having equal score are removed from the queue and added to the network iteratively, until the size limit of the network is reached.

Managing the network sizes through this $t$ parameter has the advantage that this way the ratio of the number of suboptimal alignment columns to optimal columns is uniform over the whole tree, and the memory usage can be estimated from $t$ quite accurately.

3.3 Results and discussion

We measured the accuracy of the Reticular Alignment method on BAliBASE v1.0 and v2.0 Ref1-5 datasets and compared it to the performance of the well-known alignment software ClustalW, MAFFT and FSA. See results in Fig. 3.2 and Fig. 3.3.

We were interested in how the accuracy of the RetAlign method is related to the choice of different parameters. Since the parameter space is four dimensional (guide tree building, similarity scoring, gap scoring, threshold value in the generalised Waterman-Byers algorithm) with several choices along each dimension, I only present our findings regarding the selection of the threshold value $t$ (see the next subsection).
3. RETICULAR ALIGNMENT

Figure 3.3: Comparison of alignment software on BAliBASE v1.0
Alignment accuracy of multiple alignment programs compared to that of RetAlign as measured on BAliBASE v1.0 using the provided bali_score tool (SP and TC scores are both shown). RetAlign was run with sequence weighting on, a single guide tree iteration and with a reticular threshold of 200. FSA was run in maximum sensitivity mode. MAFFT was run with the --auto switch and ClustalW with the default settings.

3.3.1 Alignment accuracy

Our heuristic method has a key parameter, \( t \), that controls how deep the search space should be, or how many sub-optimal alignments should be considered. It is not surprising that this parameter has a profound effect on the alignment accuracy and the running time of the algorithm.

As we have expected, in general, the alignment accuracy improves as the \( t \) parameter is increased (see Figure 3.4). However, we note that this effect is not always monotonous.

There can be two fundamentally different reasons for why a widened search
Figure 3.4: Dependency of alignment accuracy (all-column SP) on the reticular threshold
Alignment accuracy achieved by RetAlign for different reticular threshold values. Accuracy here is measured as the mean all-column SP score on BAliBASE v2.01 Reference sets 1–5. RetAlign was run with sequence weighting on and pairwise indel scoring [21].

may result in worse final alignments:

- the better scored alignments are less accurate (the wider search found better scored alignments, but these alignments happen to be differing more from the BAliBASE benchmark)

- because of the local decisions at the guide tree nodes it may happen that more “noise” is propagated upwards. For a detailed explanation, see Fig. 3.5.
3. RETICULAR ALIGNMENT

Figure 3.5: Explaining how the internal score might decrease with the reticular threshold.

In this particular example $s_a$ is the score of the best alignment at internal node $v_a$, with $t_1$ as the threshold value. At an upper level, namely, at node $v_b$, an $x_{b,1}$-network is generated (score: $s_{b,1}$) based on the $x_{a,1}$-network. Then the best alignment at the root has score $s_{r,1}$. After increasing the value of $t$ to $t_2$, the best alignment at node $v_a$ is still the same, but the network of suboptimal alignments to consider at higher levels is the wider $x_{a,2} > x_{a,1}$ network. From this network, it is possible to find a better scored alignment at node $v_b$, with score $s_{b,2} > s_{b,1}$. If $s_{b,2} - x_{b,2} > s_{b,1}$ then the $x_{b,2}$-network might be so different from $x_{b,1}$, that they do not contain any common alignments. Therefore, the best alignment at the root obtained from the $x_{b,2}$-network might have a score $s_{r,2} < s_{r,1}$.

3.3.2 Memory and Computational demand

As we increase the threshold value, not surprisingly, the size of the alignment network will grow, too.

Figure 3.7 shows the dependence of running time on the value of the reticular
3.3 Results and discussion

Figure 3.6: Dependency of internal score on the reticular threshold.
Best internal alignment score achieved by RetAlign for different reticular threshold values. Score is the mean internal score on BAliBASE v2.01 Reference sets 1–5. RetAlign was run with sequence weighting on and pairwise indel scoring. On average, this internal score is monotonously increasing, although we did find example sets of sequences for which the internal score decreased by increasing \( t \).

threshold. The plot shows that the empirical running time grows quadratically with the reticular threshold [21]. “This agrees well with the theoretical considerations that the time required to align two alignment networks is proportional to the product of the two network sizes. The memory usage is also quadratic with the threshold value in the current implementation (data not shown), which restricts the applicability of the software to 30–50 sequences of intermediate size (on a typical modern laptop computer) due to memory requirements, but this can be circumvented using checkpoint algorithms, see [94].”
3. RETICULAR ALIGNMENT

Figure 3.7: Execution time growth rate with the reticular threshold
Total time required to run RetAlign on BAliBASE v2.01 Reference sets 1–5 for different reticular threshold values. RetAlign was run with sequence weighting on and pairwise indel scoring.

3.3.3 Conclusions

In general, corner-cutting methods restrict the search space within the dynamic programming table for searching the best scored alignment. But previous methods still become inefficient when the number of sequences increases.

In this chapter I introduced a progressive alignment method called Reticular Alignment, which improves upon pre-existing methods by obtaining a set of optimal and suboptimal alignments at each step of the progressive alignment procedure. This set of alignments at each inner node of the guide tree is represented by a network of alignment columns, thus breaking free of the high-dimensional dynamic programming table. The alignment set usually consists of high-scored alignments that are not necessarily neighbours in the dynamic
programming table (see, for example, the already mentioned fig. 3.2 in [95]). It follows that the convex hull of the set of all these alignments represented in the high dimensional DP table contains a larger set of alignments. This way Reticular Alignment has an improved chance of finding accurate alignments without the need of more memory and running time, compared to previous algorithms [21].

Reticular Alignment’s method can also be combined with any kind of scoring scheme, so it provides a tool to assess the relative importance of sophisticated scoring schemes versus more exhaustive searches in the space of possible alignments when looking for more accurate results.

3.4 Implementation details and authors’ contributions

The paper “Reticular alignment: A progressive corner-cutting method for multiple sequence alignment” is a product of the whole team of its authors. István Miklós proposed the extension of Waterman–Byers algorithm for aligning a network of alignments to a network of alignments, and implemented a raw prototype. Adrienn Szabó (me) and Ádám Novák developed the majority of the current RetAlign implementation. Ádám Novák proposed the data structures and algorithms for efficient score calculation, and created the benchmarking framework to compare alignment programs. Adrienn Szabó tested the software on various data sets.

Source codes are available from the project webpage at http://phylogeny-cafe.elte.hu/RetAlign/
3. RETICULAR ALIGNMENT

3.4.1 Overview of the core Java classes

The core classes, representing the $x$-networks, and the algorithm of aligning networks with networks (also “flattening” or “linearizing” them) were implemented by me.

Also, the classes representing the guide trees and the two common tree building methods (UPGMA and NJ) in the `retalign.impl.tree` package were implemented and tested by me.

For a brief summary of the package hierarchy and the core classes, see Figure 3.10.

A short description of the core classes from the `retalign.impl` package:
3.4 Implementation details and authors’ contributions

**Figure 3.9:** Aligned sequences in the “Output” tab of the Reticular Alignment program.

**AlignColumn** represents an alignment column with \( c \) characters, within the network of an alignment of \( c \) sequences. It contains links to the previous and next columns, so the network can be traversed easily. The best alignment score that the column is part of is also stored, so that columns can be sorted by this property when building up a network.

**AlignNetwork** contains all the **AlignColumn** objects building up a network in a **List**. It stores the weights of sequences (used in a more sophisticated scoring scheme). A function `getBestAlignment()` is able to return the best scoring alignment path from the network.

**GuideTree** contains a pointer to the root, and couple of recursive functions that traverse a guide tree, and, for example, count the number of leaves in a subtree, find a node by name, or compute sequence weights.

**NetworkAligner** does optimized pairwise reticular alignment. It supports an arbitrary substitution matrix, gap opening, extension and tolerance parameters and allows progress monitoring. It contains the forward and backward dynamic programming tables and the functions to fill them.

**TreeNode** represents a single node of the guide tree with references to one
3. RETICULAR ALIGNMENT

Figure 3.10: Left: A screenshot of the package hierarchy from Eclipse. Right: contents of the core package, `retalign.impl`.

ancestor (except for the root) and two descendant nodes (unless the node is a leaf). It stores the branch length to the parent node and the `AlignNetwork` belonging to this node, and has methods that allow traversals.

Altogether I took part in designing the classes and contributed over 2000 lines of code to the Java sources of the RetAlign tool. I also took part in testing the software and writing the `javadoc` documentation.
Efficient representation of uncertainty in MSA

In this chapter I present a framework introduced in “Efficient representation of uncertainty in multiple sequence alignments using directed acyclic graphs” [26] and “Approximate statistical alignment by iterative sampling of substitution matrices” [96].

Note: I re-used some images and sentences (captions, etc.) of the original texts but restructured the papers for brevity, and I concentrate on the one hand on parts that are essential for understanding the key concepts; and on the other hand on my contribution. For more details, see our paper [26] and the arXiv publication [96]. I will clarify in Section 4.3 which parts of this collaboration were my work.

4.1 Introduction

The quality of a multiple sequence alignment usually has a serious impact on the results of subsequence sequence analyses (homology modeling, phylogenetic
inference, etc), so assessing an alignment’s inherent uncertainties would be often useful.

In order to represent uncertainty in alignments in a statistically coherent and efficient way, we use a special graph structure that connects alignment columns [26]:

“The alignment DAG provides a natural way to represent a distribution in the space of multiple sequence alignments, and allows for efficient computation of quantities of interest averaged over alignments, as well as summaries of the distribution.”

4.2 Methods

In this section I outline very briefly the related key ideas and concepts; for a deeper description of these refer to [26] and [96].

- **Representing the distribution of sampled alignments:** After a set of plausible alignments has been collected, a compact representation of this path distribution is a DAG. Using our framework, algorithms can be designed for efficient inference on exponentially large sets of alignments.

- **The alignment column graph and equivalence classes of nodes:** See subsection 4.2.1

- **Probability distributions on alignment DAGs:** The alignment space is high-dimensional, so each alignment will typically occur with a very low frequency. Because of this fact, the relative probabilities of entire alignments are very difficult to estimate directly by their observed frequencies. “However, a particular column may occur in many different alignments, allowing the marginal probability of each column, averaged over all alignments, to be estimated much more efficiently.” – [26]
4.2 Methods

- **Summarizing the alignment distribution:** Due to the high-dimensional and constrained nature of the state space, standard summary statistics such as the mean are not applicable in this case, so we approached the task within a decision theoretical framework, whereby the choice of summary is designed to minimize the expected value of a particular loss function, also known as the posterior risk.

- **Efficient data structures:** It is essential to ensure that the space complexity of the data structure is less than the total number of paths through the graph. The simplest way to represent a graph would be via storing a list of neighbor alignment columns for each node. However, it is enough to use the predecessor and successor equivalence classes, and this way we can further increase the space efficiency.

For exact definitions and statistical details see [26] and [96].

### 4.2.1 The alignment column graph

The alignment column graph (ACG), defined in [26], is an acyclic graph of aligned symbols, through which a path represents a valid global alignment of the corresponding sequences. A small example is depicted on Figure 4.1.

The number of possible global alignments is equal to the number of distinct paths in the ACG. This number is typically very large, growing rapidly with the number of intersection points between the alignments used to generate the graph (see Figure 4.2).

To delineate the ways in which a set of columns can be recombined to form new alignments, equivalence classes of columns have been introduced. For details, see [26].
Figure 4.1: Interchanges between alignments can result in a multiplication of the number of possible paths through the DAG. a) Two alignments and their coded versions. b) The resulting alignment DAG contains an interchange column, such that there are four paths through the DAG, arising from only two alignments. c) Correspondence between alignment columns and edges connecting cells in a dynamic programming matrix [26].
Figure 4.2: The proportion of the posterior mass contained in paths through the DAG increases rapidly with the number of samples. For the pairwise example discussed in the text, the proportion reaches in the order of 10–15% of the total posterior mass with just 100 samples, increasing to over 80% after including 2000 samples (left panel). In contrast, the proportion of posterior mass contained within the individual samples is very small (right panel) [26].
4. EFFICIENT REPRESENTATION OF UNCERTAINTY IN MSA

4.3 Implementation details and measurements

My contribution to the implementation and evaluation of our method of using alignment column graphs to represent a set of alignments included:

- Searching and selecting an appropriate data set to perform measurements on
- Writing R scripts to generate randomly perturbated substitution matrices for creating many alignments of the same sequence set
- Writing scripts to evaluate alignment accuracy
- Running WeaveAlign on OXBench data
- Generating plots of alignment scores, comparing different parameter settings and datasets
- Testing WeaveAlign, and writing a short user manual
- Reading the manuscript of the paper [26] and suggesting minor improvements.

I will extend on the first three points in the following subsections.

4.3.1 OXBench dataset and its selected subsets

OXBench – as the title of Raghava et al.’s paper [97] states – is “a benchmark for evaluation of protein multiple sequence alignment accuracy”. It includes a suite of reference alignments (created based on protein three-dimensional structures) together with evaluation measures and software that allow automatically generated alignments to be benchmarked.

We used OXBench to evaluate the correctness and accuracy of our method. Reference alignments were obtained from version 1.3 of OXBench, available at
www.compbio.dundee.ac.uk/downloads/oxbench/oxbench_1.3.tar.gz (my last access was at January 30th, 2015).

In order to assess the effect of the growing number of sequences to be aligned, while also taking into account other factors, we opted to analyze subsets of one large alignment. I chose one of the largest alignments of OXBench to carry out experiments on: this was dataset number 12 (data/align/fasta/12), including 122 proteins.

To avoid dealing with subsets containing highly similar proteins corresponding to protein subfamilies within the original set, we used a greedy algorithm to select maximally dissimilar sets of particular sizes. This way we produced sets of sequences of size 15, 33 and 60 proteins, in addition to the original, whole set of 122 sequences. These alignment samples are on GitHub at https://github.com/statalign/WeaveAlign inside of the testdata/oxbench/reference_alignments folder. An example script for computing the minimum-risk summary and alignment accuracy across the samples can also be downloaded from there.

![Diagram showing subsets of sequences](image)

**Figure 4.3:** Representation of the final subsets that we selected contain 15, 30, 60 and 122 sequences respectively. The larger ones always include all protein sequences from the smaller subsets and extend them further.

The datasets themselves, and the steps that document the selection process
are also available from a separate public code repository: https://github.com/ador/mpd-alignment-sets.

4.3.2 Perturbing substitution matrices and generating alignments

Using an approximate iterative MCMC procedure, that includes random perturbations of the scoring matrix used for alignments (described in detail in [96]), we generated 2000 alignments for each dataset.

The algorithm iterates between sampling substitution matrices, and computation of the best score-based alignment using a standard alignment software, for example MUSCLE. This gives us a set of alignments according to a simplified posterior distribution.

As we have seen in subsection 2.1.1, usually score matrices are derived from the observed mutation ratios, therefore we can think about their individual elements as numbers that represent mutation probabilities. Therefore, a perturbation procedure that simply adds a small amount of uniform random noise to the original score matrix, is a valid method for generating alternative score matrices.

I wrote the software (a small R package and scripts that use it) that carries out the uniform random perturbation of the score matrices. The amount of added noise can be adjusted via a parameter. Program codes are open sourced, available on GitHub, at https://github.com/ador/mxNoisifyPackage.

Figure 4.4 plots the scores of alignments with the randomly perturbated matrices, and the measured summary alignment scores.
4.3 Implementation details and measurements

Figure 4.4: Each box depicts the measured summary alignment score (red dot) compared to the input alignments’ scores from which set the DAG was built (aligning 60 sequences with MUSCLE, using \( N = 5000 \) randomly perturbed scoring matrices). The increasing numbers on the \( x \) axis are different values of a parameter that regulates the amount of random noise added to the original scoring matrix. As we increase the random noise, the average score of the generated individual alignments slowly decreases, but the produced summary alignments still have a consistently good score. For details, see subsection 4.3.2.

4.3.3 Evaluating alignment accuracy

AMA accuracy score, as described in subsection 2.1.5, yields a higher value for alignments that are more similar to a reference alignment. I implemented its computation and added it to the open source repository of WeaveAlign, available at https://github.com/statalign/WeaveAlign.
4. EFFICIENT REPRESENTATION OF UNCERTAINTY IN MSA

4.3.4 Automation, pipeline

I implemented an automated pipeline to generate the alignments, build up the alignment network, and then evaluate and plot the result scores. It keeps track of the parallel experiments performed over the different protein sets and alignment parameters (for example, gap opening and extension scores) by automatic naming of the generated folders and files.

An example configuration could look like the next bash code fragment:

```bash
matrixPerturbation=2.0
matrixNum=500
seedFrom=42

matrixDirEnding=rndMx-$\{matrixPerturbation\}$-n$\{matrixNum\}$-s$\{seedFrom\}$
matrixDir=$\{baseDir\}$/data/randomMatrices/$\{matrixDirEnding\}$
wvaAlignment=$\{baseDir\}$/data/wva/wva-result-$\{matrixDirEnding\}$.fsa
fastaInput=master-12.fa # simple filename, within $\{baseDir\}$/data/fasta
referenceAlignment=$\{baseDir\}$/data/fasta/master-12.aln.fsa
outJavaScoreFile=$\{baseDir\}$/data/scores/jscores-$\{matrixDirEnding\}$.csv
mafftAlignmentsDir=$\{baseDir\}$/data/mafft-out/mafftOut-$\{matrixDirEnding\}$
baseMafftAlign=$\{mafftAlignmentsDir\}$/noRnd.fsa
```

Here, the `matrixPerturbation` parameter defines the extent of perturbation of alignment score matrix (by default, the base is BLOSUM62, but could be another matrix): it will be the upper value limit that would be added to or subtracted from some values at randomly selected positions of the score matrix. `seedFrom` defines a starting random seed value; in this example 500 random matrices will be generated, each with a different random seed in the range 42–541. The remaining listed parameters set the input and output file paths.

The main steps of the pipeline can be read from the following script fragment:
4.4 Results and conclusions

I do not detail the above steps of this pipeline, they are quite straightforward, they just pass parameters to and call different tools, described elsewhere throughout this chapter (or, in case of MAFFT, in Chapter 2).

As an example of the plots generated, see Figure 4.4.

4.4 Results and conclusions

Figure 4.5 shows that running time increases linearly with the size of the input, as it was expected. (Note that the figure here includes the time for creation of the DAG structure and the execution of the minimum-risk summary algorithm, but does not include the time taken to generate the input set of alignments.)
4. EFFICIENT REPRESENTATION OF UNCERTAINTY IN MSA

**Figure 4.5:** Total runtime (black circles) and time spent creating the DAG structure (red circles) when generating a minimum-risk summary alignment, as a function of the number of input alignments. Results shown for alignments generated on 20 globin sequences, timed on a single AMD Opteron 2.3GHz core [26].

Our measurements detailed in [26] showed that: “This framework can be used to generate a statistically meaningful summary alignment from a collection of alignment samples. While other published methods generate summary alignments whose accuracy scores are often similar to, and in some cases worse than, a randomly chosen sample from the distribution, our methodology consistently yields alignments that are significantly more accurate than the majority of the alignment samples, leading to improvements in downstream tree inference.”
### Table 4.1: Some statistics and measurement results on the selected datasets with varying numbers of sequences. The posterior over alignments does not become more diffuse as the number of sequences is increased; this can be seen by the decrease in mean equivalence class size, and slight increase in the average marginal probabilities. (Timings were carried out using a single AMD Opteron 2.3GHz core.) [26]

<table>
<thead>
<tr>
<th>Number of sequences</th>
<th>15</th>
<th>33</th>
<th>60</th>
<th>122</th>
</tr>
</thead>
<tbody>
<tr>
<td>Benchmark alignment length</td>
<td>144</td>
<td>150</td>
<td>152</td>
<td>157</td>
</tr>
<tr>
<td>Mean eq. class size</td>
<td>15.2</td>
<td>11.8</td>
<td>12.4</td>
<td>11.1</td>
</tr>
<tr>
<td>Average marginal</td>
<td>0.19</td>
<td>0.21</td>
<td>0.25</td>
<td>0.23</td>
</tr>
<tr>
<td>MinRisk rank, (g = 0)</td>
<td>0.67</td>
<td>0.85</td>
<td>0.84</td>
<td>0.92</td>
</tr>
<tr>
<td>MinRisk rank, (g = 0.5)</td>
<td>0.85</td>
<td>0.95</td>
<td>0.69</td>
<td>0.74</td>
</tr>
<tr>
<td># columns in DAG</td>
<td>20288</td>
<td>26782</td>
<td>26221</td>
<td>30305</td>
</tr>
<tr>
<td>Time to read alignments (s)</td>
<td>0.5</td>
<td>0.8</td>
<td>1.2</td>
<td>2.1</td>
</tr>
<tr>
<td>Total runtime (s)</td>
<td>0.9</td>
<td>1.3</td>
<td>1.9</td>
<td>3.0</td>
</tr>
</tbody>
</table>
5

Protein Search

Searching for specific proteins in a large database still poses challenge. Despite the fact that better and better tools are constantly being developed, as the sizes of these databases grow, new, faster, and more accurate approaches are worth to be built.

In this chapter I will present an interdisciplinary project that is in-between a “traditional” data mining project and a specific branch of bioinformatics: pattern analysis and search of proteins in a large database.

5.1 Background

Alzheimer’s disease is a neurodegenerative type of dementia, a neurological disorder with memory loss and cognitive decline, caused by a growing number brain cell deaths. It typically begins with mild symptoms and slowly becomes more and more severe.

According to the estimations of The Alzheimer’s Association\(^1\) AD accounts for between 60% and 80% of all cases of dementia. The Association’s 2013

\(^1\)http://www.alz.org/
statistical report says that just over a tenth of people in the over-65 age of the population in the US is affected (around 5 million people).

The usual clinical duration of the disease is eight to ten years, but sometimes patients live up to 25 years after diagnosis. Approximately one quarter of all AD is familial (i.e., at least two persons in a family have AD), of which approximately 95% is late onset (age over 65 years) and 5% is early onset AD (age below 65 years) [98].

AD is linked to genetic disorders [98]: three forms of early-onset familial AD (EOFAD) caused by mutations in one of three genes (APP, PSEN1, PSEN2) are recognized. Bacanu et al. [99] have shown evidence for heritability of psychotic features in AD (delusions and hallucinations). Subsequent genetic studies discovered that mutations of the APP gene can cause early-onset AD with cerebral amyloid angiopathy [100].

The “product” of the APP gene, Amyloid beta A4 protein, functions as a cell surface receptor and performs physiological functions on the surface of neurons relevant to neurite growth, neuronal adhesion and axonogenesis. It is involved in cell mobility and transcription regulation through protein-protein interactions1.

A not so common amino acid duplex, HD – with presumed metal binding properties – has been found in human Alzheimer precursor protein (see Figure 5.1).

This pattern, Histidine (H) followed by Aspartic acid (D), can be found within the transmembrane, non-conserved region of the Alzheimer precursor proteins. It has been found to play active role in different enzymes’s catalytic processes, usually by metal coordination. Previous results in [100] suggest that HDs located in disordered regions in CAEDs (extracellular domains) of transmembrane proteins might have a role in facilitating or regulating inter- or

1From http://www.uniprot.org/uniprot/P05067, at 02. 10. 2015. 12:45 CET
5.1 Background

Figure 5.1: The last 300 amino acids of the 770 AA long Human Amyloid beta A4 protein (UniProd id: P05067). The HD pattern is highlighted with a green box, the transmembrane (TM) domain has a brown border, and metal binding positions are marked with light blue dots. There are 21 amino acids between HD and the transmembrane domain.

Intra-molecular interactions.

A preliminary search (carried out manually by Zoltán Zádori, virologist at MTA-ATK\textsuperscript{1}) for other relevant occurrences of the HD pattern pointed towards a yet undiscovered potential role or function of this dyad.

Because of its important role in the development of Alzheimer’s disease, we will focus our attention to proteins or protein families that exhibit similar properties to the Amyloid beta A4 protein, with regard to having a HD duplex in their CAEDs.

So, our research questions are:

1. What are the transmembrane (TM) protein groups that have a HD pattern, not too far from their TM domain, within the extracellular region?
2. Can these be considered as examples of “non-position-specific conservation” of the HD motif?

Regarding the second question, by non-position-specific conservation we mean that the occurrence of the HD pattern throughout sequence segments\footnote{Veterinary Medical Research Institute, Hungarian Academy of Sciences, Budapest, Hungary}.
of different protein groups (exhibiting similar functional properties) remains higher than what would be expected under the *null hypothesis model* that assumes *no* selection pressure to keep these HDs in the disordered regions.

The challenge here is to find these similar, but no too similar groups of protein fragments, where evolution had enough time to jumble the amino acids somewhat, but still close in terms of having a similar function. The problem is that a standard, simple sequence alignment works poorly on these disordered regions, so we need to find alternative approaches.

Given a relevant dataset, the above questions can be answered by using data mining approaches and tools, combined with pre-existing solutions from bioinformatics.

### 5.2 Brief summary of the project and its results

I developed – with the generous help of Mátyás Bachorecz and Nóra Juhász, who contributed about 25-25% of the codebase – some specific software tools for solving the tasks outlined above.

The project started at the end of 2011, when Nóra Juhász joined our research group at SZTAKI, and, with my guidance, wrote Perl scripts to count HD statistics within the SwissProt database, and started implementing the first steps of a first version of the data process pipeline (the final pipeline will be introduced in 5.4). In a couple of months we realized that the complexity of the task needs a more “manageable” language than Perl. At the end we decided to switch to Java, because of its portability, readability, speed, and object-oriented nature.

Unfortunately, due to other, time-limited projects, we had to put this HD-hunting aside for about 10 months. Then, Mátyás Bachorecz joined me to
re-implement the existing pipeline levels, and add new functionality (the tree searcher component, for example) during 2013.

The experiments that helped in fine-tuning the pipeline and choosing parameters were mostly run by me.

5.2.1 Results

On the one hand, the pipeline itself can be considered as the result of the three-year-long research process. Each level of the pipeline was run many times, with different parameters, or with different inputs from the previous levels. We experimented with other ideas and steps, but those that were not found to be useful at the end were left out from the final pipeline. We believe that these levels on top of each other can form a starting point for developing slightly different pipelines for other, similar tasks of protein search.

On the other hand, by running the pipeline with the parameters that I will describe in section 5.4, we get a handful of groups of proteins. To decide if the HD dyad within them has a common function – which could possibly mean that we found new examples of convergent evolution – more experiments have to be carried out. Unfortunately, the synthesis of proteins, and running in vitro experiments on them is both expensive and time-consuming, so it seems that we have to wait at least a year more for the biological validation of the developed pipeline.

5.3 Data

I downloaded the Swiss-Prot release 2015_04 (2015 April) of UniProtKB, that contains 548208 protein sequence entries [101, 102].

The Swiss-Prot, as many sequence databases, has two classes of data: the core sequence data, and a second one with different annotations. The following
5. PROTEIN SEARCH

summaries of metadata have been taken from the user manual of Uniprot [103].
For each sequence entry the core data consists of:

- sequence data
- citation information (bibliographical references to scientific papers)
- taxonomic data (description of the biological source of the protein)

Among others, the annotation metadata consists of the description of the following items:

- function(s) of the protein
- domains and sites, for example, calcium-binding regions, ATP-binding sites, zinc fingers, etc.
- secondary structure, e.g. alpha helix, beta sheet
- quaternary structure
- similarities to other proteins
- disease(s) associated with any number of deficiencies in the protein.

The annotations we will be interested in are ones from the “feature table” that describes functional domains, and the “keyword” lines, that include important metadata about the protein. See page 92 for an example.

5.4 Pipeline

In the following subsections I present the steps of the process that I followed to find protein families that contain significantly more of the HD dyad than can be expected, and are metal binding, and might give deeper insight into the role of the pattern related to AD. Figure 5.2 gives a visual representation of the process.
5.4 Pipeline

Swiss-Prot Protein Database (2014 04)

548k proteins

Filtering Eukaryotae TM proteins

13k proteins

(optional) Predicting TM orientations

Cutting TM + outward sections

13k fragments

Clustering protein fragments

50 clusters

Clustal omega tree building

50 trees

Selecting subtrees

Final trees with HD motif

Figure 5.2: Flow diagram of protein data (green boxes) and the major processing steps or 'levels' (purple rounded boxes).
5. PROTEIN SEARCH

All the steps were carried out by Java programs – described in Section 5.5 – called by Bash scripts. To reproduce each of these steps and their results, consult also the Appendix.

Preliminaries, general technical information and notations

In the following sections, I will use the next notations and background colors for the scripts and configuration or data files:

1. `$ command to be run`

2. `contents of a data file or settings file`

Timings

Unless noted otherwise, timings were measured on a laptop machine with Intel Core i5-3210M CPU processor (2.50GHz) and an SSD.

Running scripts

All scripts in the APP-HD-pattern-runner repository were written in a way so that in a proper environment (for example in the provided docker container) the scripts just have to be called without any parameters to reproduce the results exactly as described here in this thesis.

Optionally they take two parameters: a path to another repository (either that of ProteinPatternSearch or PhyTreeSearch, based on the part of the pipeline) and a path to a configuration file that specifies the exact behavior. Examples of such configuration files can be found in the repository, also available here: github.com/ador/APP-HD-pattern-runner/tree/master/configs

I uploaded a Docker image\(^1\) that contains all the necessary tools to “re-play” the steps to be introduced in the next subsections.

In the next subsections we will walk through the steps or levels of the

---

\(^1\)For a quick introduction to Docker, and its usage, see Appendix A.1
pipeline. These can be matched fairly well to the general data mining steps mentioned in Chapter 1. Level 0 and 1 of the pipeline correspond to data cleaning and data selection; level 2 can be seen as a specific form of transformation; levels 3 – 5 include application of a classic data mining algorithm, a standard bioinformatics tool, and a new smart heuristic solution for finding subtrees; while level 6 corresponds to knowledge presentation. To complete the process, one more step would be necessary, preceding the last one: pattern evaluation (this is work in progress).

**Downloading the data and pre-processing – level 0**

I included two scripts within the GitHub repository to download the original Swiss-Prot dataset. The first fetches the dataset version that has been used in this thesis:

```
$ get_2015_04_uniprot_dataset.sh
```

The next collects the most recent version of the Swiss-Prot proteins:

```
$ get_latest_uniprot_dataset.sh
```

Finally, the following command starts the first level of filtering the SwissProt database, by calling a script (`filter_dat_rows.sh`) and passing a path and a configuration file to it:

```
$ filter_dat_rows.sh /home/yoda/git/ProteinPatternSearch/home/yoda/.../filter_dat_rows.props
```

Let us have a look at the necessary config file (long paths have been truncated in the middle):

```
inputDatFile = /home/.../data/uniprot-rel_2015_04/uniprot_sprot.dat
outputDatFile = /home/.../uniprot_sprot_rowfiltered.dat
throwOutRows = DT,DE,OG,OH,OX,CC,DR,R
```
What happens when we run this command is that the whole Swiss-Prot dataset will be read and rows with the specified labels will be thrown out. The input data file will be reduced by 70%, so we get a 881 MB output file instead of the original 2.9 GB input. This filtering took less than two minutes.

The result data file will contain items of protein descriptions like the one shown here:

```
<table>
<thead>
<tr>
<th>ID</th>
<th>COX6C_BOVIN</th>
<th>Reviewed; 74 AA.</th>
</tr>
</thead>
<tbody>
<tr>
<td>AC</td>
<td>P04038; A0JNM4;</td>
<td></td>
</tr>
<tr>
<td>GN</td>
<td>Name=COX6C;</td>
<td></td>
</tr>
<tr>
<td>OS</td>
<td>Bos taurus (Bovine).</td>
<td></td>
</tr>
<tr>
<td>OC</td>
<td>Eukaryota; Metazoa; Chordata; Craniata; Vertebrata; Euteleostomi;</td>
<td></td>
</tr>
<tr>
<td>OC</td>
<td>Mammalia; Eutheria; Laurasiatheria; Cetartiodactyla; Ruminantia;</td>
<td></td>
</tr>
<tr>
<td>OC</td>
<td>Pecora; Bovidae; Bovinae; Bos.</td>
<td></td>
</tr>
<tr>
<td>PE</td>
<td>1: Evidence at protein level;</td>
<td></td>
</tr>
<tr>
<td>KW</td>
<td>3D-structure; Complete proteome; Direct protein sequencing;</td>
<td></td>
</tr>
<tr>
<td>KW</td>
<td>Membrane;</td>
<td></td>
</tr>
<tr>
<td>KW</td>
<td>Mitochondrion; Mitochondrion inner membrane; Reference proteome;</td>
<td></td>
</tr>
<tr>
<td>KW</td>
<td>Transmembrane; Transmembrane helix.</td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>INIT_MET 1 1 Removed. {ECO:0000269</td>
<td>PubMed:2844245,</td>
</tr>
<tr>
<td>FT</td>
<td>EC0:0000269</td>
<td>PubMed:2988583}.</td>
</tr>
<tr>
<td>FT</td>
<td>CHAIN 2 74 Cytochrome c oxidase subunit 6C.</td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>/FTId=PRG_0000191301.</td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>TOPO_DOM 2 12 Mitochondrial matrix.</td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>TRANSMEM 13 53 Helical.</td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>TOPO_DOM 54 74 Mitochondrial intermembrane.</td>
<td></td>
</tr>
<tr>
<td>FT</td>
<td>HELIX 13 39 {ECO:0000244</td>
<td>PDB:1V54}.</td>
</tr>
<tr>
<td>FT</td>
<td>HELIX 41 53 {ECO:0000244</td>
<td>PDB:1V54}.</td>
</tr>
<tr>
<td>FT</td>
<td>HELIX 57 66 {ECO:0000244</td>
<td>PDB:1V54}.</td>
</tr>
<tr>
<td>SQ</td>
<td>SEQUENCE 74 AA; 8610 MW; E0638DB1B7AE9E4C CRC64;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>MSTALAKPQM RGLLARRLRF HIVGAFMVSL GFATFYKFAV AEKRKKAYAD FYRNYDSMKD</td>
<td></td>
</tr>
<tr>
<td></td>
<td>FEEMRKAGIF QSAK</td>
<td></td>
</tr>
</tbody>
</table>
```
5.4 Pipeline

Filtering transmembrane proteins – level 1

We filtered the original protein set to keep only transmembrane proteins (that have a “Transmembrane” keyword within the annotated Swiss-Prot dataset) that belong to the sub-phylum *Eukaryotae* and consist of at least 50 amino acids.

The command to be run in the Docker container’s `APP-HD-pattern-runner` directory:

```bash
$ ./scripts/level_1/filter_dat_rows.sh
```

The corresponding configuration file has to include configuration parameters like the ones in this example:

```plaintext
inputDatFile = /home/.../uniprot_sprot_rowfiltered.dat
inputFastaFile = /home/.../uniprot_sprot.fasta
outputDatFile = /home/.../sprot_euk_min50.dat
outputFastaFile = /home/.../sprot_euk_min50.fasta
matchProteinPatterns = KW:Transmembrane|OC:Eukaryota
minSeqLength = 50
```

This way we reduced the data size to less than 100 MB, keeping only 21223 proteins, less than 4% of the original dataset.

Cutting TM and extracellular fragments – level 2

To further focus on the interesting parts, it would be desirable to select only fragments from the whole protein sequences that belong to the transmembrane part (usually about 20 amino acids), plus an outward-reaching, *extracellular* fragment of 40–70 AA (amino acids) long piece with each transmembrane domain. These fragments should then contain approximately 60–90 amino acids. Note that a protein can have multiple fragments to consider, because it can happen that a longer protein has more than one transmembrane domain (these
are called “multi-pass” proteins, see Fig. 5.3).

![Figure 5.3: A schematic image of a transmembrane protein. E = extracellular space; P = plasma membrane; I = intracellular (cytoplasmic) space. Image source: Wikipedia (http://en.wikipedia.org/wiki/Integral_membrane_protein, license: CC BY-SA 3.0).]

But we have a problem here: Swiss-Prot (as of April, 2015) contains information about the “orientation” (intra-, or extracellular) of the non-transmembrane sections in only about one third of the cases: 13198 proteins of 38276 Eukaryota transmembrane proteins.

I investigated the possibility of predicting this orientation, using external tools. The details are in Appendix A.2.

To keep going with our quest of finding transmembrane protein families with HD patterns within their extracellular parts, we have two choices:

- either discarding those proteins from the database that do not have annotated cytoplasmic/extracellular topology information, and only use about 34 % of the available Eukaryota TM proteins in the remaining part of our analysis
- or using an external tool, for example HMMTOP [104, 105] or TMHMM [106], and then sharing (making available online, for example) the resulting pre-
dictions, to allow reproducibility of the subsequent steps.

The reason for having to share HMMTOP / TMHMM results, and not including them directly into the published pipeline is the fact that the licenses of both tools disallow re-distribution.

For keeping this research easily reproducible, I chose option one.

So I changed a bit the filtering of level 1, added “FT:Extracellular”, to keep only proteins that have at least one annotated extracellular domain:

```
inputDatFile = /home/.../uniprot_sprot_rowfiltered.dat
inputFastaFile = /home/.../uniprot_sprot.fasta
outputDatFile = /home/.../sprot_euk_topology_min50.dat
outputFastaFile = /home/.../sprot_euk_topology_min50.fasta
matchProteinPatterns = KW:Transmembrane|OC:Eukaryota|FT:Extracellular
minSeqLength = 50
```

This new filtering resulted in 13198 selected proteins.

Now it's time to focus our attention to the next level filtering: selecting only “substrings” from these proteins, consisting of the transmembrane domain, and the extracellular part next to each (of length 40–70 AA).

The command to be run in the Docker container’s APP-HD-pattern-runner directory:

```
$ ./scripts/level_2/protein_fragments.sh
```

The above command will run my ProteinPatternSearch java program (for a summary see 5.5.1), and by default a configuration like this will be used:

```
inputDatFile = /home/.../sprot_euk_topology_min50.dat
inputFastaFile = /home/.../sprot_euk_topology_min50.fasta
outputFastaFile = /home/.../sprot_euk_fragments_40-70.fasta
minLenOfExtracellularPart = 40
maxLenOfExtracellularPart = 70
```
5. PROTEIN SEARCH

Some multi-pass TM proteins had more than one proper fragment; others’ fragments were too short and were discarded. After this post-filtering step we ended up having 13394 protein fragments to deal with.

Clustering protein fragments – level 3

Having still a large number of protein sequence fragments, it makes sense to use clustering to pre-organize them into homologous and thus probably functionally similar groups [107]. This way we can improve the speed, and even sensitivity of subsequent analysis - namely, finding closely related small groups with the HD pattern. I tried two fundamentally different clustering approaches, but only the second one gave meaningful end results. For details, refer to Appendix A.3.

In the Docker container’s APP-HD-pattern-runner directory, this command has to be run to perform the clustering of fragments with Weka[108]:

```bash
$ ./scripts/level_3/clustering.sh
```

Multiple alignment and tree building – level 4

Clustal Omega is a relatively new software tool that enables fast multiple alignment of many thousands of proteins [48, 49]. I used the default settings and five iterations 50 times separately to compute the 50 evolutionary trees from the clusters of protein fragments. I wrote a script to automate this process. After downloading the Clustal Omega binary (see the tools/get_clustalOmega.sh script), one can run the following script to align the sequences in each cluster and build all the evolutionary trees:

```bash
$ ./scripts/level_4/build_trees.sh <clusters dir from level 3> <level 4 output directory>
```

Within the supplied Docker image, it is enough to use the “docker auto” parameter pair if the pipeline has been run with the default settings so far:
In our case, for the 50 clusters the total running time was a bit more than one hour (67 minutes). (Note: without clustering it would have taken a couple of tens of hours to build a gigantic tree of all the 13 thousand protein fragments).

Now we have the 50 phylogenetic trees (each containing a couple of hundreds of leaves) in Newick format.

**Finding subtrees with the pattern – level 5**

Our next goal is to find those subtrees within the clusters that contain many leaves with HD pattern.

Subsection 5.5.2 contains more details of the Java software package that has been developed for this step of the data processing pipeline, here I focus only on the usage.

Similarly to the previous steps a config file has to be written that specifies the location of the input and output files, together with some parameters that define the task itself:

```bash
1 treeFilesDir = /home/.../level_4/
2 fastaFilesDir = /home/.../level_3/
3 outputFileFilesDir = /home/.../level_5/weka_c50_min7_55/
4 seqPattern = HD
5 minLeafNum = 7
6 minPatternPercent = 55
```

One should run the following script from the *APP-HD-pattern-runner* git repository:

```bash
$ ./scripts/level_5/filter_trees.sh <path to PhyTreeSarch> <config file>
```

When using Docker, it is possible to omit the parameters:

```bash
$ ./scripts/level_5/filter_trees.sh
```
5. PROTEIN SEARCH

The results we got with the configuration described above: 16 subtrees (newick and fasta files) that contain the HD motif in at least 55 percent of their leaves and have at least 7 leaves.

Altogether they contain 158 sequence fragments (leaves), of which 107 contain the HD pattern.

Visualization and analysis of results – level 6

Manual checking of over a hundred proteins is still a lot of work and very error-prone. To further aid a biologist in her investigation, I automated the process of re-aligning the sequences of the selected subtrees and visualizing the results, showing colored alignments.

```bash
$ cd scripts/level_6/
$ ./realign_subtree_seqs.sh <path to level5 "outputTreeFilesDir"> <output dir of realign>
$ ./visualize_alignments.sh <path to colorLatexAlignments> <path to output dir of realign> <output dir for .tex and pdf files>
```

Inside the supplied Docker container, it is enough to run the following script without parameters (it will call the above two scripts with pre-set paths):

```bash
$ ./realign_and_vis_docker.sh
```

One of the result sets (with filename sub14tree0) contains Amyloid Beta proteins – finding this group at the end of our analysis validates the process, at least in some extent, because these were the main cause of starting the investigation. Their alignment:

<table>
<thead>
<tr>
<th>Protein</th>
<th>Region</th>
<th>Flags</th>
<th>Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>P08592</td>
<td>630-723</td>
<td>b</td>
<td>PANTENEVEPVDAARPAAADRGLTRPGSGLTNIKT</td>
</tr>
<tr>
<td>P12023</td>
<td>630-723</td>
<td>b</td>
<td>PANTENEVEPVDAARPAAADRGLTRPGSGLTNIKT</td>
</tr>
<tr>
<td>P53601</td>
<td>630-723</td>
<td>b</td>
<td>PANTENEVEPVDAARPAAADRGLTRPGSGLTNIKT</td>
</tr>
<tr>
<td>P79307</td>
<td>630-723</td>
<td>b</td>
<td>PANTENEVEPVDAARPAAADRGLTRPGSGLTNIKT</td>
</tr>
<tr>
<td>Q51S80</td>
<td>630-723</td>
<td>b</td>
<td>PANTENEVEPVDAARPAAADRGLTRPGSGLTNIKT</td>
</tr>
<tr>
<td>Q60495</td>
<td>630-723</td>
<td>b</td>
<td>PANTENEVEPVDAARPAAADRGLTRPGSGLTNIKT</td>
</tr>
<tr>
<td>Q95241</td>
<td>611-704</td>
<td>b</td>
<td>PANTENEVEPVDAARPAAADRGLTRPGSGLTNIKT</td>
</tr>
<tr>
<td>O93279</td>
<td>599-689</td>
<td>b</td>
<td>NANTENEVEPVDAARP1PDGGLTRPVSS--LAL</td>
</tr>
</tbody>
</table>
The HD pattern is highlighted with white letters within the alignment, and the sequence fragment identifiers at the beginning of each line have the following five fields:

1. UniProt access number (for example, ‘P08592’)
2. Fragment beginning (the index of its first amino acid within the protein, starting from 1)
3. Fragment ending
4. Position code: ‘a’ means that the extracellular fragment is positioned after a TM part; ‘b’ stands here if it was found before the TM domain.
5. ‘1’ means that the sequence does contain the HD pattern; the value is ‘0’ otherwise.

The above is a very “conservative” alignment, it is very probable that these proteins have a common origin and function. To a biologist, who looks for cases of non-position-specific conservation, the interesting alignments would be less conservative, and the position of the HD pattern would not be fixed. So, let us have a look at some other alignments.\(^1\)

\(^1\)Due to space limitations, I will not list here all of the 50 resulting alignments, only three of them, that I found most interesting. The whole set of these final align-
Another result group contains glucosidases involved in the degradation of cellulose biomass, for example, proteins named “Probable glucan 1,3-beta-glucosidase D” for different organisms. Note that some of the protein fragments here contain the HD pattern twice:

<table>
<thead>
<tr>
<th>Protein</th>
<th>Start</th>
<th>End</th>
<th>HD Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1DMX4</td>
<td>307-397</td>
<td></td>
<td>WIFIGLALIIIVIVIPAVVVSKK-HDNKS--</td>
</tr>
<tr>
<td>BOY7W2</td>
<td>306-396</td>
<td></td>
<td>WIFIGLALIIIVIVIPAVVVSKK-HDNKS--</td>
</tr>
<tr>
<td>B8NNK9</td>
<td>298-388</td>
<td></td>
<td>WUIVIVLVVVLAIVIPAVVMSKK-RGDIDKKSG</td>
</tr>
<tr>
<td>Q2UMV7</td>
<td>298-388</td>
<td></td>
<td>WUIVIVLVVVLAIVIPAVVMSKK-RGDIDKKSG</td>
</tr>
<tr>
<td>Q5AVZ7</td>
<td>301-391</td>
<td></td>
<td>WIFIGLALIIIVIVIPAVVVSKK-HDNKS--</td>
</tr>
<tr>
<td>A1CTI3</td>
<td>302-392</td>
<td></td>
<td>LLIQGIVVILILIVIPAVVSSKK-HDNKPA--</td>
</tr>
<tr>
<td>Q0CHZ8</td>
<td>311-401</td>
<td></td>
<td>WIFIAVVVVVLLAIPIPAVAVSSKK-HNEKKS--</td>
</tr>
</tbody>
</table>

The last example shown here is a “typical” result, where the alignment is not so nice, and clearly, some smaller groups of closely related protein fragments were grouped together in the corresponding subtree.

<table>
<thead>
<tr>
<th>Protein</th>
<th>Start</th>
<th>End</th>
<th>HD Pattern</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q35409</td>
<td>23-114</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q27960</td>
<td>213-303</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q9DBP0</td>
<td>214-304</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q9J09</td>
<td>214-304</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q58DX5</td>
<td>122-212</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q5ZN23</td>
<td>20-110</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q9CZR2</td>
<td>8-101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

An example result is included in the GitHub repository as an example result, and can be downloaded from https://github.com/ador/APP-HD-pattern-runner/blob/master/data/example-result/HD-pattern-final_aligns.zip?raw=true.
In the above example three items are *Sodium-dependent phosphate transport protein 2Bs* (from the second to the fourth lines in each section), we have two *Glutamate carboxypeptidase 2s* (‘O35409’ and ‘Q5WN23’), and two *N-acetylated-alpha-linked acidic dipeptidase*.

As we could see from the three result examples (especially the third one depicted above), the quality of the result alignments is not ensured to be high with this procedure. Either a manual check (by a biologist) or more sophisticated statistical methods are needed to find the handful of result sets that are really valuable as candidates for being new examples of convergent evolution.

**The missing level: statistical evaluation**

To check that the results (the final subtrees) at the end of the pipeline, are indeed showing examples of non-position-specific conservation, and to prove the assumption that the high ratios of HD-containing leaves is not purely due to chance, but a product of an evolutionary selection pressure, we would need to simulate the process of evolution.

Our null hypothesis is that the disordered regions of proteins are fairly free to change, and the point mutations are independent from each other, so the
entire process can be modeled by the repeated application of an amino acid substitution matrix, with uniform choices for a position to be changed, and selecting a replacement amino acid according to the corresponding column of the substitution matrix.

If we found that a selected subtree contained significantly more HD patterns in its leaves compared to the statistical occurrence of HDs in the simulated sequences, then we could discard the null hypothesis, and suggest further investigation of the protein group to the biologist user, as a possible example of non-position-specific conservation of the HD motif.

There exist statistical methods and software packages that we could leverage for this task (for example, StatAlign [109]), but I still need to explore their applicability to this exact scenario.

### 5.5 New software packages

I developed specifically for the tasks of this project two separate tools in Java. The first, ProteinPatternSearch is a collection of basic filters and I/O classes, used throughout the data processing pipeline discussed in the previous section. The second project, PhyTreeSearch, as its name suggests, deals with the problem of finding re-occurring patterns within proteins clustered into an evolutionary tree.

Both the Java sources and the scripts and configuration files are made available in open source repositories on GitHub (github.com):

**ProteinPatternSearch**: includes Java classes that do the pre-processing steps of filtering the relevant proteins, that contain a pattern. Available at https://github.com/ador/ProteinPatternSearch.

**PhyTreeSearch**: was also written in Java, and implements subtree-searching methods. GitHub page: https://github.com/ador/PhyTreeSearch.
5.5 New software packages

Scripts and configs: a repository that contains all the necessary scripts and settings to use the previous two and conduct a filtering pipeline. Download from: https://github.com/ador/APP-HD-pattern-runner.

I also created a Docker image that includes a proper environment for these scripts and codes. I introduce its basic usage in Appendix A.1.

5.5.1 ProteinPatternSearch

This Java project\(^1\) implements a solution – sometimes leveraging external tools – for the majority of the protein search pipeline, for example filtering protein fragments, and clustering.

The tasks this software package is able to perform are:

- **COMPUTE\_STATS**: Expects a Swiss-Prot .dat file and counts 'H', 'D' and 'HD' frequencies, and prints an expected 'HD' frequency, assuming an uniform and independent random distribution of amino acids.

- **FILTER\_DAT\_ROWS**: Expects a .dat file and codes of Swiss-Prot lines to throw out; writes a shorter .dat file.

- **FASTA\_AND\_DAT\_FILTER**: Filters both a .dat and a .fasta file, keeping only proteins that match a pre-defined pattern of keywords and other features, as well as an optional minimal sequence length.

- **TM\_STATS**: Prints statistics of transmembrane proteins of a .dat file (number of total proteins, number of TM proteins, and number of TM proteins with extra- or intracellular annotation).

- **TM\_EXT\_FRAGMENTS**: Expects a .dat and a .fasta file, writes a .fasta file with extracellular protein fragments and the neighboring transmembrane sections. The minimum and maximum fragment lengths can be specified via the `minLenOfExtracellularPart` and `maxLenOfExtracellularPart` properties.

\(^1\)Source codes available at GitHub: https://github.com/ador/ProteinPatternSearch
5. PROTEIN SEARCH

**SPLIT_FASTA_TO_CLUSTERS_KCLUST** : Reads kClust output files and a .fasta input; writes many .fasta files corresponding to the clusters. Discards too small clusters, if the minClusterSize parameter is set.

**STATS_ARFF_FOR_WEKA** : Generates an .arff file (the input format of Weka) from a .fasta file of proteins or protein fragments; the attributes (columns of the .arff) will be the different amino acid counts, or amino acid group frequencies for each protein(fragment).

**WEKA_CLUSTERING** : Runs k-means clustering, built into Weka. Besides the input and output paths, mandatory parameters are: numberOfClusters and randomSeed.

**SPLIT_FASTA_TO_CLUSTERS_WEKA** : This task is very similar to the task “SPLIT_FASTA_TO_CLUSTERS_KCLUST”, but works with Weka output files instead of kClust results.

### 5.5.2 PhyTreeSearch for selecting subtrees

I developed a tool, called *PhyTreeSearch*¹, (in Java language, with the help of Mátyás Bachorecz) for automatic extraction of “interesting” subtrees from an evolutionary tree. The user can specify (by editing a simple properties file) what counts to be “interesting” in terms of:

- an amino-acid pattern occurring in leaf sequences (in our case it was “HD”)
- a minimum threshold $P$ so that only subtrees that contain at least $P$ percent of leaves matching the pattern are returned
- a minimum tree size in terms of leaf count or tree height

The program works by reading in properties file that defines the inputs, outputs, and other parameters. A sample configuration file:

---

¹The source code for the PhyTreeSearch java package can be downloaded from GitHub: https://github.com/ador/PhyTreeSearch
5.6 Further work

After reading a tree (from Newick format) and a .fasta file that contains the sequence data for the leaves of the tree, the tool builds up the Tree of TreeNode objects, and appends the fasta sequences to the leaves by matching their identifiers.

Then the main procedure traverses the tree and checks for existence of the required pattern within the sequence strings; finally a recursive method adds up at each inner node the number of its descendants that have the pre-defined pattern (HD in our case). Finally, subtrees with at least the pre-set size and having a high-enough percentage of HD sequences are written to result files (newick tree and fasta formats).

If the treeColors option is set, then the generated newick trees will have red leaves if they contain the required pattern in the corresponding protein sequence string (see Figure 5.4).

I also included a script and example data in the code repository to facilitate the ease of use.

5.6 Further work

Statistical evaluation of the results

As already mentioned in subsection 5.4, developing an automated method for testing if the prevalence of the HD motif is above what could be expected under
the null hypothesis (no selection pressure to keep at least one HD dyad), would be very useful. Then a user could loosen the constraints somewhat (allowing more subtrees to be selected even with a lower HD percentage at the leaves), and possibly find more interesting candidates without having to manually check many alignments by automatically filtering out the ones that do not match the evolutionary criteria set for non-position-specific conservation.

**Measuring the stability of the pipeline procedure**

It is also worth noting that we had to make somewhat arbitrary decisions at certain levels of the pipeline (the number of clusters that we split the whole set of fragments into, or the random seed used there). Changing these parameters could result in significantly different result subtrees at the end. Investigating the effect of these parameters is a work to be done in the future.

**User interface**

Now the user has to tinker with editing text files and scripts; but someone without a programming background would probably be more convenient with using a graphical user interface for setting the parameter values for the different levels of the pipeline.
Discussion

The science of biology offers many great opportunities to a data scientist and a software developer to take part in the challenges of making sense of the gargantuan amount of data being gathered constantly.

The large size of the datasets allows us to leverage statistical approaches and use data itself as the driving force of an investigation. Building a model that is capable of concisely describing the data and enables forecasting is not easy, but fortunately many intelligent methods are available for model building. Of course, models are never 100% accurate; very often heuristics are used to simplify a problem so that it becomes tractable.

Using heuristics

There are two different (but somewhat related) reasons for using heuristics:

• because the problem at hand is way too complex

• because we have too much data to deal with, and even moderately complex algorithms would take too much time to run.
Chapters 3 and 4 are examples of the first case: multiple sequence alignment is indeed a highly complex problem to solve. It is NP-hard [5], so using heuristics when trying to align more than a handful of sequences is unavoidable.

Finding protein families containing some pre-defined pattern, on the other hand, does not require very specific or sophisticated algorithms – but the size of the data, the need to handle hundreds of thousands of sequences does not allow an easy solution. The technics used in Chapter 5 definitely fit into our second category of heuristics.

Summary of new scientific results

Efficient corner-cutting algorithm for multiple sequence alignment: Reticular Alignment

Usually, corner-cutting methods define a compact, mostly convex part of the dynamic programming table for narrowing down the search for the best scored multiple alignment. We introduced a new progressive alignment method called Reticular Alignment, which obtains a set of optimal and suboptimal alignments at each step of the progressive alignment procedure. This set of alignments is represented by a DAG network of alignment columns.

This novel corner-cutting approach allows the efficient search of the space of multiple sequence alignments for better alignments. The method has a parameter $t$, which affects how much of the alignment space is explored. The Reticular Alignment method can be combined with any scoring scheme, and in this way, we were able to infer what is the relative importance of sophisticated scoring schemes versus more exhaustive searches in the alignment space. The conclusion is that it is important to increase the search space in order to find high-scored alignments.
The Reticular Alignment method could usually find more accurate alignments when \( t \) was increased; although we did find some interesting cases when the final MSA score decreased with a higher value of \( t \), see Figure 3.4 in Chapter 3.

We found that combining sophisticated scoring schemes with the Reticular Alignment progressive alignment approach yielded a method whose accuracy is comparable to that of cutting-edge alignment methods, Clustal, MAFFT and FSA (Figures 3.3 and 3.2).

It should be noted that this procedure of explicitly tracing the possible alignment paths – as a means of a very strict corner cutting method – can not be easily enhanced, unless some dependencies between the alignment columns would be taken into account, but that would require much more data and intelligent algorithms to be trained on it.

The software package RetAlign is available from the project’s webpage at http://phylogeny-cafe.elte.hu/RetAlign/.

A new representation for storing and carrying out statistical computations over a set of multiple sequence alignment paths

Representing a set of alignments as a DAG (directed acyclic graph) over alignment columns as nodes provides a way to quickly compute a distribution of alignment paths in the space of possible multiple sequence alignments, and other quantities of interest averaged over alignments [26].

Joining a large collection of alignment paths – in our experiments, up to a couple of thousands of them – into a DAG network allows for downstream inference to be averaged over a substantive sample of alignments. Due to interchanges and crossovers at the common alignment columns, the number of
alignments encoded in the network is usually a couple of orders of magnitude larger than the number of original alignment paths used to generate the DAG, so the effective sample size is greatly increased.

It is also possible to weight each alignment according to a more reliable estimate of the posterior probability, rather than analyzing only a small set of individual samples.

Note that many standard algorithms that were designed to work on single alignments – for example, forward-backward algorithms for HMMs – can be fairly easily adapted so that they can handle alignment DAGs as well.

In [26], we presented a general framework for dealing with alignment uncertainty: on the one hand we explained the statistical background, while on the other hand, we tested our ideas on real data by implementing the framework in Java as part of the WeaveAlign package.

A new protocol for finding re-occuring motifs in disordered regions of transmembrane proteins

Suspecting a role in the development of Alzheimer’s disease, we were looking for examples of “non-position specific conservation” of an amino acid pattern (HD) within a certain distance of transmembrane (TM) domains of TM proteins.

I worked with the Swiss-Prot database of UniProtKB, containing over half a million of protein sequences and their annotations.

The proposed protocol consists of the following, individually parameterizable steps, or levels:

1. Filtering transmembrane proteins
2. Cutting TM and extracellular fragments
3. Clustering protein fragments
4. Multiple alignment and tree building
5. Finding subtrees with the pattern
6. Visualization and analysis of results

I wrote two Java packages, the first of which (*ProteinSearch*) contains classes to handle the Swiss-Prot files and perform the filtering, cutting and clustering of proteins; while the second gives an easy-to-use solution to the problem of searching subtrees of a large evolutionary tree that adhere to some pre-defined properties (in terms of containing an amino acid pattern in the sequences, for example).

Scripts and configuration files for running the Java programs according to the above levels can be found in the code repository: https://github.com/ador/APP-HD-pattern-runner.

Figure 5.2 in Chapter 5 gives a visual overview of the process.

Reproducibility of research findings

Nowadays the “reproducibility crisis” of life sciences, and “reproducible research” (RR) are popular topics of different blogs and other internet sites.\(^1\) Not surprisingly, many of the infamous studies that had to be retracted because of unreproducible results are related to human disease studies [110, 111, 112].

The problem of un-reproducibility is not restricted to healthcare-related research. Although the reasons of it within the field of computer science are fundamentally different, the cure could be similar: requiring a more transparent research process, which means sharing every detail that matters.

As Hinsen writes in [113]: “... a scientific study must be documented to the

\(^1\)As a couple of examples:
http://reproducibleresearch.net/,
http://www.csee.wvu.edu/~xinl/source.html,
point that another researcher can follow the same steps and obtain the same results. Computational science currently falls short of this goal because the programs and the input data sets for a computation are rarely published.”

There are many different sides of this problem. Some key points of a complex solution could be:

**Raising awareness:** Luckily, more and more scientists believe that the growing level of unreproducible results is a problem that requires and deserves attention. New communities are being formed to address the problem, and they have already started developing tools that could help with this issue, usually around a programming language that is well suited to carry out and track computational experiments (for example, the R community has [https://ropensci.org](https://ropensci.org)).

**Requiring more openness:** Publishers should (and slowly they do) change their policies and accept a paper only if the necessary data and programs and scripts are published together with the text and plots describing the findings. Even new publication formats (“executable papers”) have been proposed, that would closely integrate the text describing the research finding with the input datasets and the source codes or binary programs being used. [113]

**Building tools that help the process:** Here, the key idea is automating the building blocks of the research process that are repetitive and inherently error-prone if done manually. Automation has the benefit that the process will be well documented and repeatable (at least in the same environment). Computer scientists are lucky in the sense that the entire process of their work could – in theory at least – be tracked by the computer itself that they work on. There already exist solutions for this, as an example,
I would mention *Sumatra*¹, a tool written in Python, that offers “automated tracking of scientific computations” (it is basically a lab notebook for the computer).

Unfortunately, currently doing research in a fully reproducible way – even in the field of computer science, where it should be the easiest – is quite time-consuming and the lack of incentives (not enough academic reward) plus the pressure to publish more results, more quickly, push researchers away from regularly applying the methodologies of RR.

To improve the situation over the long haul, science education should be extended to include trainings on tools that would enable researchers to do RR without very much effort.

**Software for research**

*How to create and maintain good software for research purposes?* — This question is not an easy one, and probably many different answers could be given – equally good. I will try to emphasize only some aspects of it, and the related problems that I often came across as I was doing my research related to the topics discussed in this dissertation.

**The importance of ease of use**

Biologists, or other researchers – who are usually not computer scientists or programmers at the same time – would like to take active part of the modeling process; for this, ergonomical and visual user interfaces are often needed. Even if a GUI (graphical user interface) is not provided, it is essential to attach at least a concise `README` file to a software tool that describes the problem that

¹[https://pythonhosted.org/Sumatra/](https://pythonhosted.org/Sumatra/)
6. DISCUSSION

this tool is trying to solve, and the steps to be taken to compile (if necessary) and use the software.

Dissemination of research software

Ideally, all software tools used and developed during a research process should be open software. But how should we disseminate these? Many good pieces of advice can be read in “Best Practices for Computational Science: Software Infrastructure and Environments for Reproducible and Extensible Research” by Stodden and Miguez [114].

Our Java software packages – runnable jar files and java sources – for multiple sequence alignment methods of Chapters 3 and 4 are openly available at the following URLs:

http://phylogeny-cafe.elte.hu/RetAlign/retalign-0.22a.zip
http://statalign.github.io/WeaveAlign/downloads.html
https://github.com/statalign/WeaveAlign

In Chapter 5, while implementing a pipeline for filtering protein fragments, I used two complementing methodologies / tools to ensure easy reproducibility of the final results:

- I shared my working environment through a Docker container (see subsection A.1.1 of the Appendix)

- I published all the software codes (java program sources and scripts, along with example configuration files) on GitHub as open source projects under Apache License 2.0¹ and Unlicense²:
  https://github.com/ador/APP-HD-pattern-runner
  https://github.com/ador/ProteinPatternSearch

¹http://www.apache.org/licenses/LICENSE-2.0
²http://unlicense.org
Science Code Manifesto

Nick Barnes, who has more than 20 years of experience in the software industry – as a researcher, programmer, software engineer, consultant, and manager – composed the Science Code Manifesto in 2011:

“Software is a cornerstone of science. Without software, twenty-first century science would be impossible. Without better software, science cannot progress. But the culture and institutions of science have not yet adjusted to this reality. We need to reform them to address this challenge, by adopting these five principles:

**Code** All source code written specifically to process data for a published paper must be available to the reviewers and readers of the paper.

**Copyright** The copyright ownership and license of any released source code must be clearly stated.

**Citation** Researchers who use or adapt science source code in their research must credit the code’s creators in resulting publications.

**Credit** Software contributions must be included in systems of scientific assessment, credit, and recognition.

**Curation** Source code must remain available, linked to related materials, for the useful lifetime of the publication.

Summary of Best Practices

In my view, the four most important set of advices from “Summary of best practices” by Wilson et al. [115] are:
6. DISCUSSION

• Write programs for people, not computers.
  – A program should not require its readers to hold more than a handful of facts in memory at once.
  – Make names consistent, distinctive, and meaningful.
  – Make code style and formatting consistent.

• Let the computer do the work.
  – Make the computer repeat tasks.
  – Save recent commands in a file for re-use.
  – Use a build tool to automate workflows.

• Make incremental changes.
  – Work in small steps with frequent feedback and course correction.
  – Use a version control system.
  – Put everything that has been created manually in version control.

• Don’t repeat yourself (or others).
  – Every piece of data must have a single authoritative representation in the system.
  – Modularize code rather than copying and pasting.
  – Re-use code instead of rewriting it.
Appendices
Appendix A

Details for the pipeline levels of Chapter 5

A.1 The Docker environment

Docker can be thought of as a system for creating and managing very lightweight virtual machines on Linux. All Docker containers (a container is a virtual image in its running state) use the same kernel as the host machine, so they are not as independent from the host machine as real virtual machines would be. But this enables an almost zero boot-up time, which is the greatest benefit of using Docker instead of a traditional virtualizing solution.¹

Current prerequisites for using Docker (I used Docker version 1.9) are: a Linux box with at least kernel 3.10 and an installed docker package. Installation help: docs.docker.com/engine/installation/².

¹For more details and information about the Docker ecosystem, visit the official page: www.docker.com
²Last accession time: January 11, 2016, 15.39.04 CET
A. DETAILS FOR THE PIPELINE LEVELS OF CHAPTER 5

A.1.1 Downloading Docker images and running containers for the pipeline of Chapter 5

To let everyone share their public images easily, the Docker community created the Docker Hub (https://hub.docker.com/). Anyone can create a free account and start uploading public Docker images. To download and use any of these, one does not even need to register. Based on the size of the image (these sometimes can measure up to gigabytes) the first startup of an image (that includes layered downloads) might take a longer time, but the subsequent runs should be quick.

I uploaded an image with all the software installed for running the pipeline of Chapter 5 to https://hub.docker.com/r/adorster/aszaboPhdCh5-2015/. Note that this image does not contain the SwissProt data, so you will have to down-
A.1 The Docker enviroment

load it via running a script within the Docker container.

To be able to use the image, at first one needs to download it from the Docker Hub:

```
$ docker pull adorster/aszaboPhdCh5-2015
```

To run the container and “get inside it”, issue this command:

```
$ docker run -t -i adorster/aszaboPhdCh5-2015:final /bin/bash
```

Now we are inside the running container, as a user called `yoda`. The machine name (displayed in the console after the username and a separating ‘@’ character) will be some random hexadecimal number, like `806df9a186d0`. This will identify this container, and will be important later, if you want to use this specific container again.

If you would like to exit the container, just press `Ctrl + D` inside the container’s shell. Another option is to issue a “`docker stop <container id>`” command on the host machine. It is important to note, that if you now issue another `docker run` command (even if it is exactly the same command as the previous one), then a brand new container will be spawned, which will be completely independent of the previously started one.

To resume work within a container after it has been stopped, at first you need to `start` it, then `attach` to it:

```
$ docker start 806df9a186d0
$ docker attach 806df9a186d0 # and press enter twice
```

Running the pipeline of Chapter 5

Within `yoda`’s home directory, you can find all the necessary source codes and even the compiled programs needed to run the levels 1–5 of the pipeline (for details, and commands see Chapter 5).
A.2 Predicting protein topology

In Chapter 5 between level 1 and level 2 we have the option of predicting the topology of transmembrane (TM) proteins, meaning predicting which are the extracellular parts (reaching outwards from the cell membrane) and intracellular (also called cytoplasmic) sections before and after the TM domains.

In about 34% of the cases (13198 proteins of 38276 Eukaryota transmembrane proteins according to the data set of April 2015) SwissProt does contain information about the “orientation” of the transmembrane proteins: “Extracellular” and “Cytoplasmic” topological domains are annotated, consecutive to the TM part. But to process the remaining 66% of the proteins, I needed an automated way to predict this kind of topology.

To investigate the available solutions, I searched the literature and the internet in general for “transmembrane protein prediction”.

I found that most of the tools for transmembrane helix prediction are made available as a web service – it is not surprising in bioinformatics, because most users (biologists) usually are curious only about a handful of proteins, and are happier to use an intuitive GUI than a programmable API. However, due to the huge number of proteins that I wanted to make predictions about, copy-pasting the sequences one-by-one into a web browser was not a viable option. So, I was looking for tools that can be run in a standalone mode locally, and found these to be promising:

HMMTOP2 (http://www.enzim.hu/hmmtop/) “is an automatic server for predicting transmembrane helices and topology of proteins, developed by G.E. Tusnády, at the Institute of Enzymology.”[104, 105]

MEMSAT-SVM (http://bioinf.cs.ucl.ac.uk/psipred/?memsatsvm=1) “is a SVM (support vector machines) based TM (Transmembrane Protein) protein topology predictor.”[116, 117]
A.2 Predicting protein topology

<table>
<thead>
<tr>
<th>Method</th>
<th>Qok</th>
<th>Qhtm1</th>
<th>Qhtm2</th>
<th>HbDiff</th>
<th>QHb</th>
<th>Gauss QHb</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMSAT-SVM</td>
<td>77</td>
<td>91</td>
<td>98</td>
<td>2.3</td>
<td>84</td>
<td>62</td>
</tr>
<tr>
<td>HMMTOP2</td>
<td>69</td>
<td>91</td>
<td>96</td>
<td>3.3</td>
<td>75</td>
<td>47</td>
</tr>
<tr>
<td>TMHMM2</td>
<td>65</td>
<td>88</td>
<td>98</td>
<td>3.2</td>
<td>74</td>
<td>47</td>
</tr>
</tbody>
</table>

Table A.1: Topography scores of the three selected methods (evaluating predictions of membrane helices versus not membrane helix)

TMHMM (http://www.cbs.dtu.dk/services/TMHMM/) is for prediction of transmembrane helices in proteins, it works as described in A hidden Markov model for predicting transmembrane helices in protein sequences[106].

A.2.1 Benchmarking

To get an overview of their performance I used a comparison tool available at The University of Sidney: http://sydney.edu.au/pharmacy/sbio/software/TMH_benchmark.shtml The benchmarking site has some options and settings, of which I used the following parameters (if a parameter/option is not not mentioned here, then I used the default value):

- Similarity is measured by local alignment (’water’) instead of the default global (’needle’) setting.

- I raised the default maximum similarities level from 30% to be 80%.

- I selected the Eukaryota kingdom as the set to run the measurements on.

For a summary of the input options of the benchmarking (and a screenshot of the settings of the benchmarking page) see Figure A.2.

Tables A.1 and A.2, and Figure A.3 show the results of the benchmarking (the meaning of the columns will be detailed below).
A. DETAILS FOR THE PIPELINE LEVELS OF CHAPTER 5

**Figure A.2:** Settings for benchmarking topology prediction (screenshot)

**Topography scores**

*Topography scores* are for scoring predictions of membrane helices versus not membrane helix.

**Qok:** Percentage of protein sequences for which all membrane helices are predicted correctly

**Qhtm1:** Percentage of all observed membrane helices that are predicted correctly (sensitivity)

**Qhtm2:** Percentage of all predicted membrane helices that are predicted correctly (specificity)

**HbDiff:** Average helix boundary position difference in residues for the prediction versus the observed helix boundaries
A.2 Predicting protein topology

<table>
<thead>
<tr>
<th>Method</th>
<th>Qok3</th>
<th>Nterm</th>
<th>ioSegQ2</th>
<th>Qio</th>
<th>Q3</th>
<th>ioResQ2</th>
<th>ioMCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>MEMSAT-SVM</td>
<td>78</td>
<td>86</td>
<td>87</td>
<td>85</td>
<td>82</td>
<td>80</td>
<td>0.56</td>
</tr>
<tr>
<td>HMMTOP2</td>
<td>66</td>
<td>81</td>
<td>81</td>
<td>85</td>
<td>61</td>
<td>50</td>
<td>-0.20</td>
</tr>
<tr>
<td>TMHMM2</td>
<td>60</td>
<td>75</td>
<td>76</td>
<td>81</td>
<td>75</td>
<td>67</td>
<td>0.42</td>
</tr>
</tbody>
</table>

Table A.2: Topology scores of the three selected methods (evaluating predictions of inside/outside topology)

**QHb:** Percentage of all observed membrane helix boundaries (2 per helix) that are predicted correctly (within distance from observed boundary)

**Gauss QHb:** Scaled percentage of all observed membrane helix boundaries (2 per helix) that are predicted correctly (score is scaled as a Gaussian curve normal distribution (mean = 0, variance = std-dev2 = 5 or chosen by user) around the observed helix boundary because observed helix boundary may not be exact)

**Topology scores**

*Topology scores* are for scoring predictions of inside/outside topology.

**Qok3:** Percentage of protein sequences for which all inside, outside and membrane helix topologies are predicted correctly

**Nterm:** Percentage of protein sequences for which the topology of the N-terminal topology is predicted correctly

**ioSeg Q2:** Percentage of correctly predicted topology segments in two-states: inside side / outside side of membrane

**Qio:** Percentage of all observed non-membrane topologies (inside side or outside side of membrane) that are predicted correctly

----

1Each sequence contributes equally so that a topology prediction that starts correctly and finishes incorrectly due to a missed membrane segment prediction rather than due to an incorrect topology prediction start will be penalised as an incompletely correct topology prediction (or not completely incorrect prediction) and long sequences will not be overly penalised in the final score.
**A. DETAILS FOR THE PIPELINE LEVELS OF CHAPTER 5**

**Figure A.3:** Benchmark results

**Q3:** Percentage of correctly predicted residues in three-states: membrane helix / inside non-membrane residue / outside non-membrane residue\(^1\)

**ioRes Q2:** Percentage of correctly predicted residues in two-states: inside non-membrane residue / outside non-membrane residue\(^1\). (each sequence contributes equally so that long sequences do not dominate the score.

\(^{1}\)Each sequence contributes equally so that long sequences do not dominate the score.
A.2 Predicting protein topology

tributes equally so that long sequences do not dominate the score)

io MCC: Matthews Correlation Co-efficient\[118\] for topology prediction of residues as either on the inside vs. the outside side of the membrane

A.2.2 HMMTOP

After requesting a license\(^1\), I downloaded and unpacked the .tgz file. The core of the tool is a single, 1100 lines long C code. The README file contains 6 lines only, but tells exactly what to do to compile a runnable binary. Only a C compiler (\texttt{cc}) is needed. HMMTOP has built-in help pages, and altogether I found it easy to use and fast.

Unfortunately its license does not allow re-distribution:

“All information, data and files are copyright[ed]. HMMTOP is produced in the Institute of Enzymology, Budapest, Hungary. There are no restrictions on its use by non-profit institutions as long as its content is in no way modified and this statement is not removed from files. Usage by and for commercial entities requires a license agreement (send an email to hmmtop at enzim dot hu).”

A.2.3 MEMSAT-SVM

After downloading and unpacking the tool\(^2\), at first I opened the README file. It consists of more than 300 lines, quite extensive, but a bit intimidating as well. It lists all the necessary external tools that are needed. Compiling seemed to be all right (with the \texttt{make} build tool).

\(^1\)A HMMTOP license for version 2.1 was requested by me via www.enzim.hu/hmmtop/ on the 12th of September, 2012

\(^2\)MEMSAT-SVM is downloadable from http://bioinfadmin.cs.ucl.ac.uk/downloads/memsat-svm/
A. DETAILS FOR THE PIPELINE LEVELS OF CHAPTER 5

Unfortunately, I run into problems when trying to fulfill the external tool requirements. MEMSAT-SVM uses a fairly old, now unsupported version of BLAST. It was not easy, but I managed to find and compile a BLAST of the required version.

I tried to run the Perl script that orchestrates the MEMSAT-runs, but even after hours of tinkering with it (during which I had to add some sleep() commands here and there to let the script run further — probably it was only tested on much older and slower hardware), I could not get past this error message:

[makemat] FATAL ERROR: Unable to recover checkpoint from
<path-to-memsat>/memsat-svm/output/memsat-svm_tmp.chk

I did search for solutions on the Web, but without any success.

A.2.4 TMHMM

TMHMM\(^1\) is somewhere inbetween the previous two tools in terms of ease of use. The README provides enough information to launch the program, describes the dependencies well, but is less than a hundred lines long, and is easy to read.

It runs at a comparable (almost the same) speed as HMMTOP, and also gives comparable results.

The problem here is the same: license. The fifth line of the README says:

\[ \text{****** PLEASE DO NOT REDISTRIBUTE THE PACKAGE ******} \]

\(^1\)TMHMM-2.0 can be obtained from http://www.cbs.dtu.dk/cgi-bin/nph-sw_request?tmhmm after accepting its terms of use and the license.
A.2 Predicting protein topology

A.2.5 Summary

As we can see from Tables A.1 and A.2, and Fig. A.3, choosing a “best method” for protein topology prediction is not very easy, different measures give different outcomes. My next planned step was to download and install all three tools and perform some measurements on the SwissProt database to get more targeted information about their performance. But then I realized that because of legal or compilation issues, I will not be able to use any of these tool in my public pipeline.
A.3 Clustering protein fragments

At first I used \textit{kClust} as a tool to cluster the fragments. I found it easy to use, and it worked quite fast. It is freely available\footnote{Download from \url{ftp://toolkit.lmb.uni-muenchen.de/pub/kClust}. Last accessed: 27th April, 2015, 16:04 CET} under GPL license. From Hauser, Mayer, and Söding’s abstract \cite{Hauser2011}: “\textit{kClust} owes its speed and sensitivity to an alignment-free prefilter that calculates the cumulative score of all similar 6-mers between pairs of sequences, and to a dynamic programming algorithm that operates on pairs of similar 4-mers. \[\ldots\] \textit{kClust} is two to three orders of magnitude faster than clustering based on NCBI BLAST, and on multidomain sequences of 20\%–30\% maximum pairwise sequence identity it achieves comparable sensitivity and a lower false discovery rate.”

Running \textit{kClust} with the default settings on my dataset of about 13400 fairly short protein fragments took only 9 seconds on my laptop. This first run resulted in 4195 clusters, of which the largest contained 87 sequences, and 28 clusters contained at least 30 fragments, and there were more than 1800 clusters with only 1 item in them. To cluster the fragments a bit more closely together, I tried to tweak the input parameters, but I did not succeed in significantly improving the results (increasing the cluster sizes).

To solve this problem, I extended \textit{ProteinPatternSearch’s} capabilities (I wrote Java code) to post-process the result files of \textit{kClust} and gain separate \textit{fasta} files for each cluster. Additionally, a \texttt{minClusterSize} parameter can be used to discard clusters that are too small – in our case I used 8 as the minimum cluster size.

The results: \textit{kClust} created 4195 clusters of the 13394 fragments, of which only 320 contained at least 8 elements, with a total of 4772 protein fragments being kept from (a bit more than a third of them).

Our next task will be to align these protein parts and build 320 trees,
A.3 Clustering protein fragments

grasp the evolutionary relationships between them.

Unfortunately, after completing the pipeline, I noticed that because of the too small clusters, only the very closely related proteins are clustered together, and this fact prevents us from discovering examples of non-position specific conservation.

Using another approach for clustering

Because of the problem of having too small clusters, an alignment-based clustering method (like kClust) is not suitable for our purposes. I developed and used a simple, new tool to cluster the fragments. It computes amino acid frequencies and uses Weka\cite{108} to perform \textit{k-means} clustering.\footnote{For details, see the https://github.com/ador/ProteinPatternSearch code repository, and within: java/src/main/java/protka/main/WekaClustering.java}

At the end, I used a properties file like this has to be put together to run the third level of the pipeline:

```
1 inputFastaFile = /home/.../sprot_euk_fragments_40-70.fasta
2 proteinStatsArffFile = /home/.../euk_fragments.arff
3 numberOfClusters = 50
4 randomSeed = 42
5 wekaOutClusterPath = /home/.../euk_weka_50_clusters_s_42.clu
6 outputFastaFilesPrefix = /home/.../fasta/cluster_50_s42_
```

The result fasta files will be named \texttt{cluster\_50\_s42\_n.fasta} where \texttt{n} will go from 0 to the number of clusters that reach the minimum size minus one.

In the Docker container’s \texttt{APP-HD-pattern-runner} directory, this command has to be run to perform the clustering of fragments with \textit{Weka}, and write the corresponding fasta files to the specified directory:

```
1 $ ./scripts/level_3/clustering.sh
```

Note that \textit{k-means} is sensitive to the initial, random placement of cluster centers, so changing the random seed, the resulting clusters can be somewhat

\footnote{For details, see the https://github.com/ador/ProteinPatternSearch code repository, and within: java/src/main/java/protka/main/WekaClustering.java}
A. DETAILS FOR THE PIPELINE LEVELS OF CHAPTER 5

different, and this might influence even the final results.
List of Figures

2.1 Groups of amino acids by physical/chemical properties. Source: Alignment III – PAM Matrices from Introduction to Bioinformatics (COMS 544) course at Iowa State University 2014. www.cs.iastate.edu/~cs544/Lectures/PAM_matrices.ppt at Jan 18. 7:40 PM, 2015 .......................... 9

2.2 PAM250 matrix from Dayhoff et al.’s publication [9] ................. 13

2.3 BLOSUM62 matrix from Henikoff and Henikoff’s publication [10] 15

2.4 Optimal alignment path within the dynamic programming table. Source of image: http://blog.thegrandlocus.com/2014/06/once-upon-a-blast at 03. 02. 2015 11.20.45 CET ................................. 23

2.5 Possible last steps in a dynamic programming table: match, insertion or deletion. Source of image: http://blog.thegrandlocus.com/2014/06/once-upon-a-blast at 03. 02. 2015 11.20.45 CET ........................... 24

2.6 Possible steps in a dynamic programming table (left) and the filled table (right). Source of image: http://blog.thegrandlocus.com/2014/06/once-upon-a-blast at 03. 02. 2015 11.20.45 CET ........................... 25

2.7 Smith–Waterman example dynamic programming table. Source of image: http://blog.thegrandlocus.com/2014/06/once-upon-a-blast at 03. 02. 2015 11.20.45 CET ................................. 26
2.8 Example where three different MSAs are produced from three pairwise alignments, but none of them is consistent with all of the pairwise cases. ................................. 36

2.9 Clustal Omega colored alignment output ................................. 39

2.10 Limitations of MAFFT: it becomes slow if the sequences are too distant or extended swaps had occurred between them .......... 40

2.11 The three main stages of the MUSCLE algorithm: Stage 1 (draft progressive), Stage 2 (improved progressive) and Stage 3 (refinement) [53] ................................................................. 41

2.12 Example ................................................................. 44

3.1 A small example of a reticular alignment network: it shows three different alignments of the sequences ALLGVGQ and AVGQ .. 48

3.2 Comparison of alignment software on BAliBASE v2.01 Refs 1-5. Alignment accuracy of multiple alignment programs compared to that of RetAlign as measured on BAliBASE v2.01 Reference sets 1-5 using the provided bali_score tool (SP and TC scores calculated on all of the columns versus on columns containing features are all shown). RetAlign was run with sequence weighting on, a single guide tree iteration and with a reticular threshold of 800. FSA was run in maximum sensitivity mode. MAFFT was run with the -auto switch and ClustalW with the default settings. 49
3.3 Comparison of alignment software on BAliBASE v1.0 Alignment accuracy of multiple alignment programs compared to that of RetAlign as measured on BAliBASE v1.0 using the provided bali_score tool (SP and TC scores are both shown). RetAlign was run with sequence weighting on, a single guide tree iteration and with a reticular threshold of 200. FSA was run in maximum sensitivity mode. MAFFT was run with the --auto switch and ClustalW with the default settings.

3.4 Dependency of alignment accuracy (all-column SP) on the reticular threshold Alignment accuracy achieved by RetAlign for different reticular threshold values. Accuracy here is measured as the mean all-column SP score on BAliBASE v2.01 Reference sets 1–5. RetAlign was run with sequence weighting on and pairwise indel scoring [21].

3.5 Explaining how the internal score might decrease with the reticular threshold. In this particular example $s_a$ is the score of the best alignment at internal node $v_a$, with $t_1$ as the threshold value. At an upper level, namely, at node $v_b$, an $x_{b,1}$-network is generated (score: $s_{b,1}$) based on the $x_{a,1}$-network. Then the best alignment at the root has score $s_{r,1}$. After increasing the value of $t$ to $t_2$, the best alignment at node $v_a$ is still the same, but the network of suboptimal alignments to consider at higher levels is the wider $x_{a,2} > x_{a,1}$ network. From this network, it is possible to find a better scored alignment at node $v_b$, with score $s_{b,2} > s_{b,1}$. If $s_{b,2} - x_{b,2} > s_{b,1}$ then the $x_{b,2}$-network might be so different from $x_{b,1}$, that they do not contain any common alignments. Therefore, the best alignment at the root obtained from the $x_{b,2}$-network might have a score $s_{r,2} < s_{r,1}$.
3.6 **Dependency of internal score on the reticular threshold.** Best internal alignment score achieved by RetAlign for different reticular threshold values. Score is the mean internal score on BAliBASE v2.01 Reference sets 1–5. RetAlign was run with sequence weighting on and pairwise indel scoring. On average, this internal score is monotonously increasing, although we did find example sets of sequences for which the internal score decreased by increasing $t$. 63

3.7 **Execution time growth rate with the reticular threshold.** Total time required to run RetAlign on BAliBASE v2.01 Reference sets 1–5 for different reticular threshold values. RetAlign was run with sequence weighting on and pairwise indel scoring. 64

3.8 A screenshot of the graphical user interface of the Reticular Alignment program. 66

3.9 Aligned sequences in the “Output” tab of the Reticular Alignment program. 67

3.10 Left: A screenshot of the package hierarchy from Eclipse. Right: contents of the core package, `retalign.impl`. 68

4.1 Interchanges between alignments can result in a multiplication of the number of possible paths through the DAG. a) Two alignments and their coded versions. b) The resulting alignment DAG contains an interchange column, such that there are four paths through the DAG, arising from only two alignments. c) Correspondence between alignment columns and edges connecting cells in a dynamic programming matrix [26]. 72
4.2 The proportion of the posterior mass contained in paths through
the DAG increases rapidly with the number of samples. For the
pairwise example discussed in the text, the proportion reaches
in the order of 10–15% of the total posterior mass with just 100
samples, increasing to over 80% after including 2000 samples (left
panel). In contrast, the proportion of posterior mass contained
within the individual samples is very small (right panel) [26]. . . 73

4.3 Representation of the final subsets that we selected contain 15,
30, 60 and 122 sequences respectively. The larger ones always
include all protein sequences from the smaller subsets and extend
them further. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 75

4.4 Each box depicts the measured summary alignment score (red
dot) compared to the input alignments’ scores from which set
the DAG was built (aligning 60 sequences with MUSCLE, using
$N = 5000$ randomly perturbed scoring matrices). The increasing
numbers on the $x$ axis are different values of a parameter
that regulates the amount of random noise added to the original
scoring matrix. As we increase the random noise, the average
score of the generated individual alignments slowly decreases,
but the produced summary alignments still have a consistently
good score. For details, see subsection 4.3.2. . . . . . . . . . . . . 77

4.5 Total runtime (black circles) and time spent creating the DAG
structure (red circles) when generating a minimum-risk summary
alignment, as a function of the number of input alignments. Re-
results shown for alignments generated on 20 globin sequences,
timed on a single AMD Opteron 2.3GHz core [26]. . . . . . . . . 80
5.1 The last 300 amino acids of the 770 AA long Human Amyloid beta A4 protein (UniProd id: P05067). The HD pattern is highlighted with a green box, the transmembrane (TM) domain has a brown border, and metal binding positions are marked with light blue dots. There are 21 amino acids between HD and the transmembrane domain. ........................................ 85

5.2 Flow diagram of protein data (green boxes) and the major processing steps or 'levels' (purple rounded boxes). ................. 89


5.4 Two example output subtrees: black leaves do not contain the HD pattern, while red leaf labels show a sequence fragment with the HD motif. ................................................................. 106

A.1 Screenshot of the Docker Registry homepage, showing some public images ................................................................. 120

A.2 Settings for benchmarking topology prediction (screenshot) . . . . 124

A.3 Benchmark results ............................................................... 126
References


REFERENCES

[22] KNOTH RENNB. JENG ROSTK EMN. An ite-
ratative method for faster sum of pairs multiple
sequence alignment. Bioinformatics, 16(9):808–
814, 2000. 28, 29, 30
DUNBRACK. Large Scale Comparison of Protein
Sequence Alignment Algorithms With Struc-
[24] SCHWARTZ A.S., MYERS E.W., AND PACHTER L. Align-
ment metric accuracy. arXiv, q-bio/0510052,
2005. 31, 32
[25] M. BLANCHETTE, W. J. KENT, C. RHEISER, L. ELINTSKI,
A. F. SHIT, K. M. ROSSIK, R. BAERINGL, K. ROSEN-
BLASS, H. CLAMSON, E. D. GREES, D. HAUSELER, AND
W. MILLER. Aligning multiple genomic sequences
with the threaded blockset aligner. Genome Res.,
[26] JOSEPH L. HERMAN, ÁDÁM NOVÁK, RUNE LINNGO, ADRIEN
SZÁRO, ISTVÁN MIKLÓS, AND JOYCE HENN. Efficient rep-
resentation of uncertainty in multiple sequence
alignments using directed acyclic graphs. BMC
Bioinformatics, 16(1), 2015. 32, 42, 43, 69, 70, 71,
72, 73, 74, 80, 81, 109, 110, 136, 137
[27] ISTVÁN MIKLÓS. Statisztikus szekvencia illesztés. PhD
thesis. ELTE TTK Növényrendszeri és Ökológiai
Tanácsék, 2002. 33
[28] J. SPOUGE. Speeding up dynamic programming
algorithms for finding optimal lattice paths.
SIAM Journal on Applied Mathematics, 49(5):1552–
1566, 1989. 33
[29] H CABRILLO AND D LIPMAN. The multiple sequence
alignment problem in biology. SIAM Journal on
multiple sequence alignment. PNAS, 86:4412–
4415, 1989. 34
[31] S GUPTA, J KECECIOGLU, AND A SCHÄFFER. Improving
the practical space and time efficiency of the
shortest-paths approach to sum-of-pairs multi-
472, 1995. 34
[32] H CABRILLO AND D LIPMAN. The Multiple Sequence
Alignment Problem in Biology. SIAM Journal on
[33] DAVID J LIPMAN, STEPHEN F ALTSCHUL, AND JOHN D KE-
CECIOGLU. A tool for multiple sequence align-
ment. Proceedings of the National Academy of Scien-
ces, 86(12):4412–4415, 1989. 35
[34] DF FENG AND RF DOOLITTLE. PROGRESSIVE
SEQUENCE ALIGNMENT AS A PRERQ-
UISITE TO CORRECT PHYLOGENETIC
TREES. Journal of Molecular Evolution, 25(4):351–
360, 1987. 35
[35] F. HOGERSWEG AND B. HESPER. The alignment of sets
of sequences and the construction of phyletic
[36] DESMOND G. HOGGINS AND PAVL M. SHARP. CLUSTAL:
a package for performing multiple sequence
alignment on a microcomputer. Gene, 73(1):237
– 244, 1988. 35, 38
[37] CÉDRIC NOTREBASQUE, DESMOND G HOGGIN, AND JAAP
HERINGA. T-coffee: a novel method for fast and
accurate multiple sequence alignment. Journal
[38] F. S. PAIS, P. D. E. C. RUY, G. OLIVEIRA, AND R. S. COSTA-
BRA. Assessing the efficiency of multiple se-
quence alignment programs. Algorithms Mol Biol,
9(1-4), 2014. 35, 38
[39] D F FENG AND R F DOOLITTLE. Progressive sequence
alignment as a prerequisite to correct phylo-
[40] DAVID W. MOUNT. Bioinformatics: sequence and genome
37, 45
[41] P.H.A. SNEATH AND R.R. SOKAL. Numerical Taxonomy,
Francisco, 1973. 37
[42] N. SATO AND M. NEL. The neighbor-joining
method: a new method for reconstructing phylo-
1987. 37
[43] MAKOTO HIROSAWA, YASUSHI TOTOKI, MASAKI HOSHIDA, AND
MASATO ISHIKAWA. Comprehensive study on itera-
tive algorithms of multiple sequence alignment.
Computer applications in the biosciences (CABIOS),
[44] A. E. TORDA. Not assessing the efficiency of mul-
tiple sequence alignment programs. Algorithms
Mol Biol, 9:18, 2014. 38


Functional Role Precursor Protein Orthologues Suggests a Stationary Selection of an HD Motif on Alzheimer

Joseph L. Herman, Adrienn Szabó, István Miklós, and István Miklós and Zoltán Zadori

REFERENCES


[103] UniProt Consortium et al. User Manual for UniProt Knowledgebase, Swiss-Protein Knowledgebase, TrEMBL Protein Database. 2015. 88


REFERENCES


Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other Hungarian or foreign examination board.

The thesis work was conducted from 2008 to 2015 under the supervision of István Miklós and András A. Benczúr at Eötvös Loránd University, Budapest.

Budapest, 2016. 26. 01.
Thesis summary

After quickly introducing bioinformatics and data mining in the first chapter, Chapter 2 explores previously existing solutions and techniques for aligning proteins and searching protein databases, by reviewing the related scientific literature.

Chapter 3 and Chapter 4 introduce new results in the topic of multiple sequence alignment: of the two, the first describes how we implemented a new kind of corner-cutting method (a progressive corner cutting multiple alignment method called “Reticular Alignment”); while the second introduces a new representation and framework (“WeaveAlign”) for storing and carrying out statistical computations efficiently over a large set of multiple sequence alignment paths.

Chapter 5 shows how we tackled the problem of finding a needle in a haystack within a biological context: we developed a framework within which a pipeline can be defined (each of the steps parameterizable) to find protein families exhibiting some pre-defined complex features in a large protein database.

The last chapter summarizes the results and includes advice on how to write software for research purposes and how to do computational research in a reproducible way.
Összefoglalás

A modern biológia tudománya egyre inkább elképzelhetetlen komoly számítógépes támogatás nélkül. A gén- és fehérjeszekvenálás egyre olcsóbbá válásával ezen adatok olyan sebességgel keletkeznek, hogy feldolgozásukhoz mindenképpen szükség van hatékony algoritmusokra és azokat implementáló programokra.

A disszertáció két nagyobb témakört ölelt fel a bioinformatikán belül: többszörös szekvenciaillesztés, illetve speciális keresés nagy fehérje-adatbázisban.

A dolgozatom első, bevezető fejezetében nagy vonalakban ismertettem a terület alapfogalmait (bioinformatika, adatbányászat), majd a másodikban a kapcsolódó eddigi kutatási eredmények irodalmát.


Az ötödik fejezetben egy új protokollt állítottunk össze, amely segítségével nem-pozíciós specifikusan konzerválódott mintázatokkal rendelkező fehérjecsorsportokat kereshetünk egy nagy adatbázisban.
Végül a záró fejezet összefoglalta az új eredményeket, és röviden kitekintett a reprodukálható kutatás-fejlesztés módszertanába.
I. A doktori értekezés adatai
A szerző neve: Szabó Adrienn
MTMT-azonosító: 10017801
A doktori értekezés címe és alcíme: Approaches to Efficient Multiple Sequence Alignment and Protein Search
DOI-azonosító: 10.15476/ELTE.2016.008
A doktori iskola neve: ELTE Informatika Doktori Iskola
A doktori iskolán belüli doktordiploma program neve: Az informatika alapjai és módszertana
A témavezető neve és tudományos fokozata: Miklós István, PhD
A témavezető munkahelye: Rényi Alfréd Matematikai Kutatóintézet

II. Nyilatkozatok
1. A doktori értekezés szerzőjeként
   a) hozzájárulok, hogy a doktori fokozat megszerzését követően a doktori értekezésem és a tézisek nyilvánosságára kerüljenek az ELTE Digitális Intézményi Tudástatban. Felhatalmazom az ELTE Informatika Doktori Iskola hivatalának ügyintézőjét, Riz-Herczeg Krisztinát, hogy az értekezést és a téziseket feltöltse az ELTE Digitális Intézményi Tudástatba, és ennek során kitöltsse a feltüntetéshez szükséges nyilatkozatokat.
   b) kérem, hogy a mellékelte kérelmenben részletezett szabadalmi, illetőleg oltalmi bejelentés közzétételéig a doktori értekezést ne bocsássák nyilvánosságára az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástatban;
   c) kérem, hogy a nemzetbiztonsági okból minősített adatot tartalmazó doktori értekezést a minősítés (dátum)-ig tartó időtartama alatt ne bocsássák nyilvánosságára az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástatban;
   d) kérem, hogy a mű kiadására vonatkozó mellékelt kiadó szerződésre tekintettel a doktori értekezést a könyv megjelenéséig ne bocsássák nyilvánosságára az Egyetemi Könyvtárban, és az ELTE Digitális Intézményi Tudástatban csak a könyv bibliográfiai adatait tegyék közzé. Ha a könyv a fokozatszerzést követően egy évig nem jelenik meg, hozzájárulok, hogy a doktori értekezésem és a tézisek nyilvánosságára kerüljenek az Egyetemi Könyvtárban és az ELTE Digitális Intézményi Tudástatban.

2. A doktori értekezés szerzőjeként kijelentem, hogy
   a) az ELTE Digitális Intézményi Tudástatba feltöltendő doktori értekezés és a tézisek saját eredeti, önálló szellemi munkám és legjobb tudomáson szerint nem sértem vele senki szerzői jogait;
   b) a doktori értekezés és a tézisek nyomtatott változatai és az elektronikus adathordozón benyújtott tartalmak (szöveg és ábrák) mindenben megegyeznek.

3. A doktori értekezés szerzőjeként hozzájárulok a doktori értekezés és a tézisek szövegének plágiumkereső adatbázisba helyezéséhez és plágiumellenőrző vizsgálatok lefuttatásához.


a doktori értekezés szerzőjének aláírása