UNIVERSITÉ LIBRE DE BRUXELLES
Interuniversity Institute of Bioinformatics in Brussels

Improving the Needleman-Wunsch algorithm with the DynaMine predictor

Olivier Boes

Advisors:
Tom Lenaerts
Wim Vranken
Elisa Cilia

Dissertation submitted in partial fullfillment
of the requirements for the degree of
Master in Bioinformatics

Academic year **2013–2014**

Acknowledgements

Besides my three advisors, I would like to collectively thank all the other bioinformatics students of this year for their patience and enthusiasm when answering my numerous naive questions about biology. As the only student this year having zero biological background, it was very helpful for me to be surrounded by students willing to exchange some of their biological knowledge with some of my mathematical and computational knowledge.

Contents

Introduction

Protein sequence alignment is, in bioinformatics, a task which aims to identify the functional, structural, or evolutionary relationships among a set of proteins believed to be related in some way (for example, proteins sharing a common ancestor). More precisely, it attempts to explain differences in proteins by finding the most likely substitutions, insertions, or deletions of amino acids residues. This is done by inserting gaps in each protein sequence, so that the gapped sequences can be represented as rows in a matrix, with the matrix columns containing residues that are either identical or similar. Such matrix is what we call an *alignment*.

Biologists have been comparing related proteins for a long time, but the earliest use of computer-based approaches can be traced back to at least 1966, with the works of Fitch [Fit66]. Since then, numerous sequence alignment algorithms have been developed, many of which being used everyday in modern bioinformatics. The most famous of these algorithms is probably the one of Needleman and Wunsch [NW70], which, although it originated in 1970, is still applied today. Furthermore, its simplicity and historical importance allowed it to become a standard introduction to many sequence alignment courses.

A protein is more than a linear sequence of amino acids: it also has a three-dimensional *structure* which is responsible for most of the protein's biological function. For the majority of proteins, this structure is unknown; but if the structure is known, even partially, it can be used to produce alignments which are biologically more accurate than alignments made by using the residue sequence alone.

In this thesis, we shall use a backbone flexibility predictor called DynaMine to acquire some additional information on a protein's structure, and we will try to use that information for improving the Needleman-Wunsch algorithm. Classical uses of Needleman-Wunsch uses a 20×20 matrix containing scores for each residue substitution. Using DynaMine and reference alignments in the BAliBASE benchmark database, we will create matrices for scoring matchings between DynaMine values. These matrices will be combined with the classical residue substitution matrices, producing Needleman-Wunsch algorithms which align sequences by using both DynaMine values and amino acid residues.

We organized the thesis in the following manner. First there is the obligatory background chapter containing all the theory necessary for the later experiments. In particular, the

Needleman-Wunsch and matrices-generating algorithms are described in details, and generalized so that we can use them with the values produced by DynaMine. The second chapter is the preliminary to our experiments: it will explain our objective, our choice of software, and how our dataset was obtained and preprocessed. Finally, there is the chapter containing the actual experiments that were conducted: it includes the creation of DynaMine scoring matrices, their combination with classical substitution matrices, the analysis of the alignments obtained using our modified Needleman-Wunsch algorithm, and a list of alternative methods that we could not further investigate because of time and scope constraints. We conclude the chapter with a summary of what was learned when writing this thesis, as well as some self-criticism. For the ones interested in computer programming, we included an appendix explaining where to find the (generalized) Needleman-Wunsch implementation used in our experiments.

Chapter 1

Background

1.1 Sequence alignment

Our main object of interest in this thesis will be the ordered *sequence* of amino acids along a protein backbone. We will begin by explaining how these sequences are encoded as strings of letters, and, most importantly, what is an *alignment* of sequences. Some basic concepts relating to proteins will be recalled, but we will not try to give an introductory course to molecular biology. Readers unfamiliar with the subject and willing to learn more can use classic textbooks such as *Campbell Biology* [CR$^+$13] and *Molecular Biology of the Gene* [WB$^+$13], but in any case, no advanced biological knowledge is required for understanding the experiments conducted in this thesis. On the other hand, some familiarity with the more computational side of things (algorithms, mathematical notation) is assumed.

The protein alphabet

Before giving the list of amino acids appearing in proteins, let us first recall some very basic concepts of cell biology. Inside every cell lies molecules called deoxyribonucleic acids (DNA) which encode the genetic instructions required for its development and functioning. DNA consists of two complementary strands built from only 4 different simpler units called the nucleotides: adenine (`A`), cytosine (`C`), guanine (`G`), and thymine (`T`). Therefore, DNA is an example (certainly the most important one) of a *biological sequence*, and can be represented using a long string of letters on the `ACGT` alphabet.

The cell uses the information contained in the DNA to assemble the molecules responsible for most biological mechanisms: the proteins. More precisely, when a protein is produced in a cell, a particular segment of DNA (called a *gene*) is first copied into another molecule called a ribonucleic acid (RNA), through a process called the *transcription*. The chemical structure of a RNA molecule is very similar to that of DNA: the main differences are that RNA is single-stranded, uses ribose sugar for its backbone (rather than deoxyribose), and the thymine nucleotide is replaced by an uracil (`U`)

nucleotide. Therefore RNA is also a biological sequence, which uses the 4-letters alphabet `ACGU`. Once produced, the messenger RNA molecule is then read by a ribosome – those are complex protein-building molecular machines found in all living cells – in order to perform the *translation*: the nucleotide sequence is decoded into an amino acid sequence, and a protein is produced. The set of rules for translating codons (triplets of nucleotides) into amino acids is called the *genetic code*, and stays the same across almost all organisms. The whole process we just described, and which could be summarized as 'DNA makes RNA makes proteins', is known as the central dogma of molecular biology.

The previous short description is of course a simplification. In reality, many other biosynthetic mechanisms and subtleties can and do comes into play, for example post-translational modification of proteins is possible (e.g. once out of the ribosome proteins can be cut in smaller pieces or conversely assembled together, or some of their amino acids can be converted to other ones), and slight variations on the genetic code can in fact occur inside the same cell (e.g. the mitochondrial code has small differences with the standard genetic code). But as we said earlier, our aim here is not to give a cell biology course.

So, since proteins are sequences of amino acids, what are the amino acids possibly present in a protein? What will be our alphabet? It is generally considered that there are 20 *standard* amino acids: their names and letter codes are listed in figure 1.1.1. However, this is in fact a bit more complicated than that: some proteins also use two additional amino acids, namely selenocysteine (`U`) and pyrrolysine (`O`). But these two are special, as they are not coded for directly in the genetic code; for example, on a messenger RNA the `UGA` and `UAG` codons, which are normally stop codons, can under very specific circumstances act as selenocysteine or pyrrolysine codons respectively [BBC$^+$91,SJK02]. Moreover, these 21st and 22nd amino acids are rare: selenocysteine is only found in 25 human proteins [KCN$^+$03], and pyrrolysine-containing proteins apparently mostly occur in organisms of the Archeae domain of life. There is also N-Formylmethionine which is the first encoded amino acid in the biosynthesis of proteins in bacteria, mitochondria or choloroplasts, but it is then often removed posttranslationaly [SST85].

In this thesis we focus on the standard 20-letters protein alphabet of figure 1.1.1, but our experiments can be quite easily applied using an extended alphabet. Of course, there exists also many *non-proteinogenic* amino acids, but since they are not found in proteins, they will be of no interest in the context of this thesis.

`A`	Alanine	`L`	Leucine
`R`	Arginine	`K`	Lysine
`N`	Asparagine	`M`	Methionine
`D`	Aspartic Acid	`F`	Phenylalanine
`C`	Cysteine	`P`	Proline
`Q`	Glutamine	`S`	Serine
`E`	Glutamic Acid	`T`	Threonine
`G`	Glycine	`W`	Tryptophan
`H`	Histidine	`Y`	Tyrosine
`I`	Isoleucine	`V`	Valine

Figure 1.1.1: *The standard protein alphabet.*

With this alphabet, and the convention that a protein sequence should be read from its N-terminal end to its C-terminal end, any protein can now be represented as a sequence of letters. For example, α-amanitin, one of the proteins responsible for the toxicity of the infamous *Amanita phalloides* mushroom, has sequence `IWGIGCNP`. This is a very small protein (an example of an oligopeptide), but most protein sequences are much longer than that: according to [Sch08], human proteins have a median size of 341 amino acid residues, and the largest one is a muscle protein with a length of 33 423 residues.

Definition of an alignment

Given a set of sequences (we are mostly interested in protein sequences but what follows can apply to any sequences of symbols), the sequence alignment task consists in the insertion of *gaps* (usually noted with the symbol '`-`') between consecutive residues of each sequence, such that 1) all gapped sequences have the same length and 2) if we write the gapped sequences in rows (thus forming a matrix), then residues belonging to a same column are similar. When using the word 'residue', we always mean an element of the sequence (an amino acid in the case of proteins): gaps are never called residues. We also allows the insertion of gaps before or after a whole sequence; those are called *end gaps*.

What 'similar' means, as well as the penalty for inserting 'too many' gaps, must be defined using a *scoring system*. A scoring system can be seen as a function assigning a *score* to every possible alignment of the given set of sequences. The job of an *alignment algorithm* is then to find an alignment with maximum score (or, in the case of approximation algorithms, an alignment with a 'good-enough' score).

There is no better way to explain what is a sequence alignment than showing one. Therefore, we collected a few sequences of our choice on the UniProtKB [Con14] protein database, and aligned them using the online Clustal Omega [SWD$^+$11, GML$^+$10] multiple sequence alignment program. We tried to find proteins related to the toxicity

of the *Amanita* genus of mushrooms, mainly because they are generally very short and we want to be able to fit the alignment on the page! In the set of sequences we also included one completely unrelated protein (from a trypanosome), to see what the aligner will do with it. The sequence alignment computed by Clustal Omega is shown in figure 1.1.2: we will first discuss it 'naively', by trying to guess what the aligner (or rather, its developers) wanted to do.

Identifiers	**Aligned sequences**	**Organisms**
D6CFW3	MSDINATRLPI--W------GIG-CDPCIGDDVTALLTRGEASLC	*Amanita phalloides*
D6CFW5	MSDINATRLPA--W------LVD-C-PCVGDDINRLLTRGENSLC	*Amanita virosa*
A8W7M7	MSDINATRLPA--W------LVD-C-PCVGDDVNRLLTRGESL-C	*Amanita bisporigera*
S4WL84	----------I--W------GIG-CNPCVGDEVTALLTRGEA---	*Amanita fuligineoides*
U5L3J5	MSDINTARLPV--F------SLPVFFPFVSDDIQAVLTRGESL-C	*Amanita exitialis*
H2E7Q5	MFDTNATRLPI--W------GIG-CNPWTAEHVDQTLASGNDI-C	*Galerina marginata*
Q04078	MAPRSLYLLAVLLFSANLFAGVGFAAAAEGPEDKGL---------	*Trypanosoma brucei*

Figure 1.1.2: *An example of sequence alignment. The five first proteins are toxins found in poisonous mushrooms of the genus* Amanita*; the sixth one is a similar protein but coming from another genus of mushrooms. The last sequence is totally unrelated to the others: it is a protein produced by the parasite which causes the African trypanosomiasis disease, or* sleeping sickness*. The first column is the sequence identifier in the UniProtKB database.*

First observation: the aligner kind of ignored our 'orphan' trypanosome sequence. It did not insert any gaps inside it (only end gaps), and its inclusion just forced the presence of two gaps common to all the mushroom protein sequences. So the aligner recognized that this sequence was very different than the others, and instead it focused on aligning the other more similar sequences. From now on we will also ignore the trypanosome sequence: it was just to show what happens when trying to align many similar proteins together with one intruder protein.

Looking at the alignment again, it is clear that Clustal Omega tries to align identical amino acids. Unsurprisingly, identical amino acids are considered 'similar', and any biological alignment algorithm will try to do its best to get them in common columns. But that is not all: even when different amino acids are aligned, we can see some pattern. For example, it seems that valine (`V`), leucine (`L`), and isoleucine (`I`) often appear in the same column: if we look at the columns containing at least one of these amino acids, we can count 13`V`, 17`L`, 14`I`, but only 11 other amino acids (residues in the unrelated trypanosome sequence were ignored). So the aligner seems to enjoy matching these three amino acids together. In fact, valine, isoleucine, and leucine, form what are called the branched-chain amino acids (BCAA), something we will no try to explain here as we are not biologists (instead see [PKMH00] and [Pát07]). But what is important is that amino acids which are distinct, but still share a common chemical property, also tends to be aligned together, at least in our example.

Therefore a biological alignment algorithm's goal is not just making 'good-looking' alignments; rather it tries to produce alignments which are 'good' *in a biological sense*.

Modifications of the nucleotide sequences in a genome (the total genetic material carried by an organism's cells) can happen: for example because of genetic recombination during reproduction, or because of mutations resulting from damage to DNA. In particular, *insertion*, *deletion*, and *substitution* of nucleotides are events which sequence alignments try to detect. Suppose for example that a part of a gene is changed from `TGCGACCCGTGC` to `TGCCCATGC`. One way of aligning these two sequences is as follows:

```
T G C G A C C C G T G C
T G - - - C C C A T G C
```

In which case the meaning of the alignment is that one deletion of `CGA` and one `G`→`A` substitution transformed the first sequence into the second (if we suppose that the second sequence is the 'original' one, then we will rather talk of one insertion of `CGA` and one `A`→`G` substitution). Since DNA contains the information for producing proteins, these substitutions and *indels* (combinations of insertions and deletions) of nucleotides translate to substitutions and indels of amino acids in proteins. In our example, if we look at a DNA codon table, it could mean that the **`CDPC`** protein subsequence was changed to **`CPC`**: there was a deletion of the **`D`** amino acid, but no amino acid substitution because both `CCG` and `CCA` codons translate to the **`P`** amino acid.

So, to summarize: if the proteins in an alignment share a common ancestor, mismatches can be interpreted as substitutions and gaps as indels introduced at some points during their evolutionary history. Moreover, the presence of highly conserved regions in a protein sequence alignment (see figure 1.1.2 for example) may suggest that these regions have some biologically important function.

Types of protein alignments

There exists different kinds of protein alignments. We give here a short summary of their most important differences, but in this thesis we will focus on only one kind of alignments: pairwise global sequence alignments.

Local and global alignments.

Global alignment means aligning every residue in every sequence, and is most used for sets of sequences that are roughly similar and of equal size. Local alignment is more useful for dissimilar sequences of different sizes, but containing smaller regions of similarity.

Pairwise and multiple alignments.

When there is only two sequences to align we speak of pairwise sequence alignment; if there are more, we say multiple sequence alignment. Aligning a large number of sequences together is generally much more complicated than aligning only two, and often requires to first compute pairwise alignments for each pair of sequences.

Structural alignments.

Something very important to know about proteins is that they are more than linear sequences of amino acids: they have 3D geometric structures, from which comes most of their biological functions. Much of this structure depends on the residue sequence: the chemical properties (e.g. polarity, charge, hydrophobicity) of the different amino acids force the protein to *fold* in a specific way (see 1.1.3 for an example). Thus a protein should not be understood as a linear 1D molecule (a common analogy is that of magnetized beads on a string). When this structure is known (but for this, we need experimental methods for structure resolution, such as X-ray Crystallography or Nuclear Magnetic Resonance), it can be used for alignments: in this case we do not want to align similar residues, rather we want to align structurally similar parts of the protein. This kind of alignment is not always possible, as the structure of proteins is not always known. In fact, as of 2014, the UniProtKB/TrEMBL protein sequence database [Con14] contains almost 80 millions entries, while the PDB protein structure database [BWF$^+$00] contains around 100 000 protein structures. But when structural alignment is possible, it gives rise to more biologically relevant alignments, as protein structure is believed to be more conserved than protein sequence [IAE09].

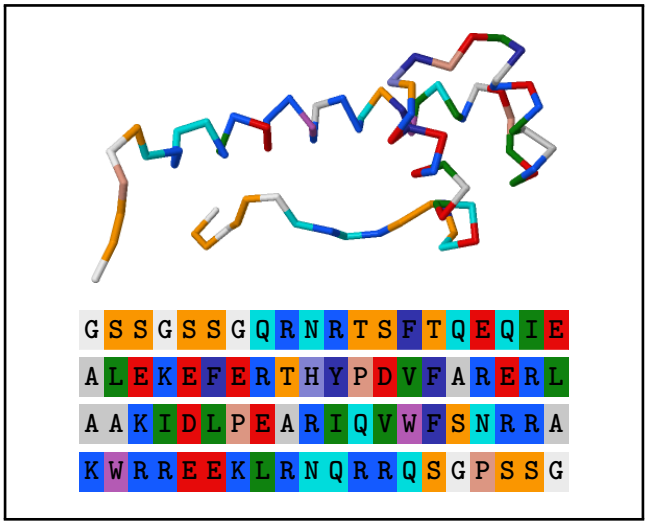

Figure 1.1.3: *3D shape of the backbone of a folded protein, together with its residue sequence. This protein has identifier* 2CUE *on the PDB database.*

Applications of sequence alignment

We would like to end this first section by listing some of the possible application of sequence alignment. This list is by no means exhaustive and we only give a concise description of each possible application; the interested reader can learn more by referring to the cited books and articles.

Sequence identification.

This is the most obvious one: if I give you a (fragment of) biological sequence, from which DNA, RNA, or protein does it comes from? Biological sequence databases such as BLAST [AGM+90] use local alignment algorithms to match a sequence query to their database, and will give you a list of the most similar sequences found. In the same way, sequence alignment can help you find the locus of a gene in a genome.

Comparative modeling.

The huge gap between known protein sequences and known protein structures was already mentioned previously. Experimental resolution of a protein structure is expensive, so another approach is to align the protein to a 'template' protein whose structure is already known, and then try to guess the unknown structure using this alignment. This approach is also known as *homology modeling* [OA12].

Protein function prediction.

This is a corollary to the previous point, since the biological function of a protein comes from its structure. Once the structure is known, many further applications become possible, such as the prediction of protein-protein interactions [Fu04], or the design of protein-binding ligands.

Phylogenetics.

Another obvious application of biological sequence alignment is phylogenetics, or the study of evolutionary relationships among groups of organisms. For example, a multiple alignment of sequences coming from different organisms can serves as a guide to the construction of phylogenetic tree [DHH11].

Genome assembly.

Current technology does not allow for sequencing a whole DNA molecule in one go. Rather, smaller overlapping DNA sequences are read and then assembled together. Sequence alignment is used to align and merge these small DNA fragments. This application was especially important for the completion of the Human Genome Project [SSHJ93].

Motif discovery.

A *motif* is a nucleotide pattern which is widespread across a genome and has a biological function, for example it could be a region of DNA to which the RNA polymerase enzyme binds before initiating a gene transcription (such a region is called a gene promoter). Alignment algorithms can be used to search these motifs, and are thus useful in gene discovery [Bin06].

Applications outside biology.

Biological sequences are not the only objects that can be aligned. Alignments algorithms have also been used for speech recognition [SC78] and computational linguistics [Mit05].

1.2 Predicting protein flexibility with DynaMine

We already explained in the previous section (see figure 1.1.3) that proteins have a 3D structure (also called a *conformation*). The distinction between four levels of structure is usually made, with the fourth level only present in the case of proteins composed of multiple subunit proteins assembled together.

primary structure:	the linear chain of amino acids (the residue sequence)
secondary structure:	the helices, sheets, and other regular shapes along the chain
tertiary structure:	the manner in which the chain fold in compact 3D structures
quaternary structure:	the arrangement of multiple folded chains fitting together

But what really interests us here is not so much the protein structure, but the possible alteration of this structure.

Protein dynamics

Besides the protein structure, there is also the protein *dynamics*. Indeed, the structure is not unique and fixed; in fact it is instable and conformational change is possible because of *flexibility* in some parts of the protein backbone [ROS+04]. There is for example the case of intrinsically disordered proteins [DLB+01, Tom02], which lack a native structure (we say that they have a *random-coil* conformation), although they could acquire one when interacting with a partner protein, forming a multi-component complex that do not fold correctly in the absence of other components [JS13]. It is possible to investigate conformational fluctuations of proteins using Nuclear Magnetic Resonance techniques [IT00, Kay98]; but these experimental methods will not be described here as it would fall outside the scope of our subject.

The DynaMine predictor

DynaMine is a predictor of protein backbone flexibility [CPT+13] that was developed at the $(IB)^2$ institute [`ibsquare.be`]; a Web server [CPT+14] for using the predictor has been set up on [`dynamine.ibsquare.be`]. DynaMine takes a protein sequence as input, and returns a corresponding sequence of numbers in $[0, 1]$ estimating the protein backbone flexibility at each residue position. More exactly, these numbers are S^2 order parameters: their definition is somewhat technical, so we prefer to point the reader to the [LS82] and [SGK96] articles. But the meaning of the S^2 order parameters is simple: a value of 0 is for very high flexibility (fully random bond vector movement) while a value of 1 is for very low flexibility (stable conformation).

Measuring S^2 parameters requires NMR, so the DynaMine predictor used the NMR data in the BMRB database [MUB+08], to which it applied the RCI predictor [BW07] to get a benchmark database of order parameters. DynaMine then uses a linear regression algorithm for making its predictions, with the context of each residue taken in consideration (the 25 preceding residues and the 25 following residues). Because of that, DynaMine should not be used on short sequences.

Figure 1.2.1 is an example of plot produced with the DynaMine Web server, for the TSP9 protein (UniProtKB identifier: I6Y9K3).

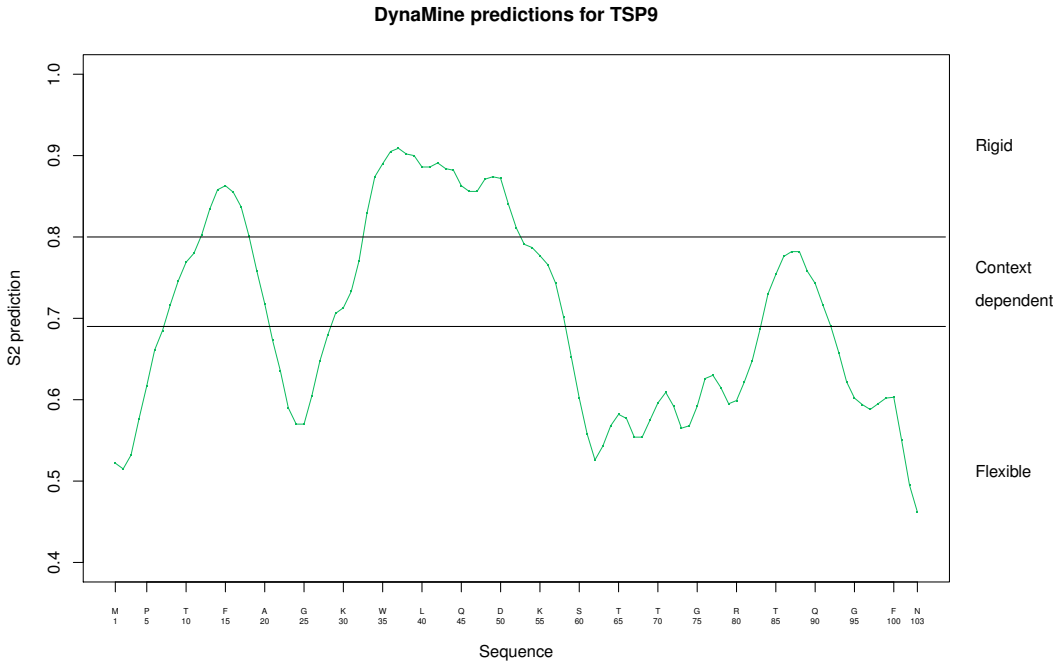

Figure 1.2.1: *A protein containing disordered regions.*

1.3 Needleman-Wunsch algorithm

For the remaining of this work we will be concerned with global pairwise sequence alignment. In the bioinformatics community, the most famous algorithm for this task is generally called the Needleman-Wunsch algorithm, although it would maybe be more correct to call it the Needleman-Wunsch-Gotoh algorithm. It is an *optimal* algorithm, which means that it produces the best possible solution with respect to the chosen scoring system. There exists also non-optimal alignment algorithms, most notably the heuristic methods used by the BLAST [AGM$^+$90,Mad13] and FASTA [LP85,LP88] softwares. Although non-optimal, these methods are faster and better suited for querying large biological databases (they were developed for this purpose). Other non-optimal algorithms which deserve to be mentioned are those using probabilistic models, in particular Hidden Markov Models [E$^+$95]. But in our case, the Needleman-Wunsch algorithm will suffice, because we do not plan to compute multiple alignments, nor will we work with extremely long sequences (such as whole genomes).

Dynamic Programming

The Needleman-Wunsch algorithm uses a *dynamic programming* method. These methods were popularized by Richard Bellman in the late 1950s when working on optimization problems for the RAND corporation [Bel52, Bel54, BD62], but the term is difficult to define precisely. In fact, as Bellman himself explains in his autobiography [Bel84]:

> *[Dynamic] also has a very interesting property as an adjective, and that is it's impossible to use the word dynamic in a pejorative sense. Try thinking of some combination that will possibly give it a pejorative meaning. It's impossible. Thus, I thought dynamic programming was a good name.*

However, maybe a possible definition would be to say that dynamic programming solves a problem by recursively breaking it into smaller subproblems, although it is more or less the same thing as what is usually called *divide and conquer* algorithms. In all cases, the word *programming* does not refer to computer programming: rather it should be understood as a synonym of mathematical optimization (like in *integer programming* or *linear programming*).

But not matter the definition, in our case, the Needleman-Wunsch algorithm will indeed compute an optimal alignment of sequences by recursively computing optimal subalignments of subsequences. Some notation will clarify what we mean. Suppose we want to align the two sequences $\mathbf{x} := (x_1, \ldots, x_m)$ and $\mathbf{y} := (y_1, \ldots, y_n)$. An alignment between the subsequences $(x_1, \ldots, x_i)$ and $(y_1, \ldots, y_j)$ is called a (i, j)-subalignment, and its maximum possible score is noted $S(i, j)$:

$$S(i,j) \quad := \quad \text{max score of all } (i,j)\text{-subalignments}$$

Once the Needleman-Wunsch algorithm will have filled the *dynamic array* S with *partial scores*, the global best score will be $S(m, n)$, and we will backtrack from it down to $S(0, 0)$ to find an optimal global alignment (which is not unique in general).

The 'score' of an alignment still has to be defined, so that a recursion relation for computing $S(i, j)$ may be derived. We will begin with the scoring system most commonly used when introducing the Needleman-Wunsch algorithm: substitution scores for matched residues and linear gap penalties. Although Needleman and Wunsch already discussed this scoring system in their 1970 article [NW70], the form in which it is now most commonly presented is due to Gotoh [Got82] (who is also responsible for the affine gap penalties version of the algorithm). An alignment algorithm very similar to Needleman-Wunsch, but developed for speech recognition, was also independently described by Vintsyuk in 1968 [Vin68]. Another early author interested in the subject is Sellers [Sel74], who described in 1974 an alignment algorithm minimizing sequence distance rather than maximizing sequence similarity; however Smith and Waterman (two authors famous for the algorithm bearing their name) proved in 1981 that both procedures are equivalent [SWF81]. Therefore it is clear that there are many classic papers, often a bit old, describing Needleman-Wunsch and its variants using different mathematical notations. For writing this section we mainly used the textbook [DEKM98].

Basic Needleman-Wunsch

For each pair of symbols (x_i, y_j) we define a *substitution score* $\text{sub}(x_i, y_j)$. This should be a good (large) score when x_i and y_j are similar and a bad (small, or even negative) score when they are dissimilar. We also define a *gap penalty*, a constant number which should be nonpositive (otherwise the algorithm will just try to add gaps everywhere!). This scoring system allows us to assign a score to each column of a pairwise alignment, and the global alignment score will be the sum of the column scores. As a basic example, let us consider

$$\text{sub}(x_i, y_j) := \begin{cases} +2 & \text{if } x_i = y_j \\ -1 & \text{if } x_i \neq y_j \end{cases} \quad \text{and} \quad \text{gap} := -1.$$

So that a match gives 2 points, but a mismatch or a gap gives a -1 penalty. On figure 1.3.1 below are two examples of alignments between sequences $\mathbf{x} =$ (CYSTEINE) and $\mathbf{y} =$ (GLYCINE), with their columns scores and alignments scores computed.

C	-	Y	S	T	E	I	N	E	
G	L	Y	-	C	-	I	N	E	
−1	−1	+2	−1	−1	−1	+2	+2	+2	= **3**

C	Y	S	T	E	I	N	E	
-	G	L	Y	C	I	N	E	
−1	−1	−1	−1	−1	+2	+2	+2	= **1**

Figure 1.3.1: *How to compute alignment scores.*

Now, remark that a (i, j)-subalignment is always of one the following forms:

- a concatenation of a $(i-1, j)$-subalignment with a column $\begin{bmatrix} x_i \\ - \end{bmatrix}$,
- a concatenation of a $(i, j-1)$-subalignment with a column $\begin{bmatrix} - \\ y_j \end{bmatrix}$,
- a concatenation of a $(i-1, j-1)$-subalignment with a column $\begin{bmatrix} x_i \\ y_j \end{bmatrix}$.

Therefore it is clear that a (i, j)-subalignment maximum score is:

$$S(i,j) \;=\; \max \begin{cases} S(i-1,\ j\ \ \ ) \;+\; \text{gap} \\ S(i\ \ \ ,\ j-1) \;+\; \text{gap} \\ S(i-1,\ j-1) \;+\; \text{sub}(x_i, y_j) \end{cases}$$

We set $S(0,0) := 0$ as a starting value (an 'empty alignment' is worth 0 points), and for simplication we also set $S(i,j) := -\infty$ whenever i or j is a negative number. This recurrence relation allows us to easily compute the best score $S(m,n)$: we just have to fill the array starting from $S(0,0)$ (for example, row by row).

Once the dynamic array is filled, we can stop there if we are just interested in the best score, but if we want to compute an optimal alignment, we have to backtrack from $S(m,n)$ to $S(0,0)$; although the procedure is relatively straighforward, we described the backtracking algorithm in more details in figure 1.3.2.

Input: dynamic array S, sequences $\mathbf{x}$ and $\mathbf{y}$

Output: optimal alignment A

- $A := \begin{bmatrix} & \end{bmatrix}$ (empty alignment)
- $(i, j) := (m, n)$
- **while** $(i, j) \neq (0, 0)$ **:**
    - **choose** (u, v) **among :**
        - $(1, 0)$ **if** $S(i, j) = S(i-1,\ j\ ) + \text{gap}$
        - $(0, 1)$ **if** $S(i, j) = S(i\ ,\ j-1) + \text{gap}$
        - $(1, 1)$ **if** $S(i, j) = S(i-1,\ j-1) + \text{sub}(x_i, y_j)$
    - **if** $(u, v) = (1, 0)$ **:** $A := \begin{bmatrix} x_i \\ - \end{bmatrix} + A$
    - **if** $(u, v) = (0, 1)$ **:** $A := \begin{bmatrix} - \\ y_j \end{bmatrix} + A$
    - **if** $(u, v) = (1, 1)$ **:** $A := \begin{bmatrix} x_i \\ y_j \end{bmatrix} + A$
    - $(i, j) := (i, j) - (u, v)$
- **return** A

Figure 1.3.2: *The backtracking part of Needleman-Wunsch.*

Multiple time in this algorithm, we have to make a choice between 1 and 3 pairs (u, v) (they correspond to the direction in which to continue the backtracking). These choices are up to you: they will all yield an alignment with the same score (recall that the optimal alignment is not unique in general).

Remark that it would be simpler, and slightly more efficient, to keep track of backpointers when filling the dynamic array, rather than 'rediscovering' the path like it is done in the above algorithm. Figure 1.3.3 gives an example of a filled dynamic array, along with its backpointers, for the alignment of `CYSTEINE` and `GLYCINE`.

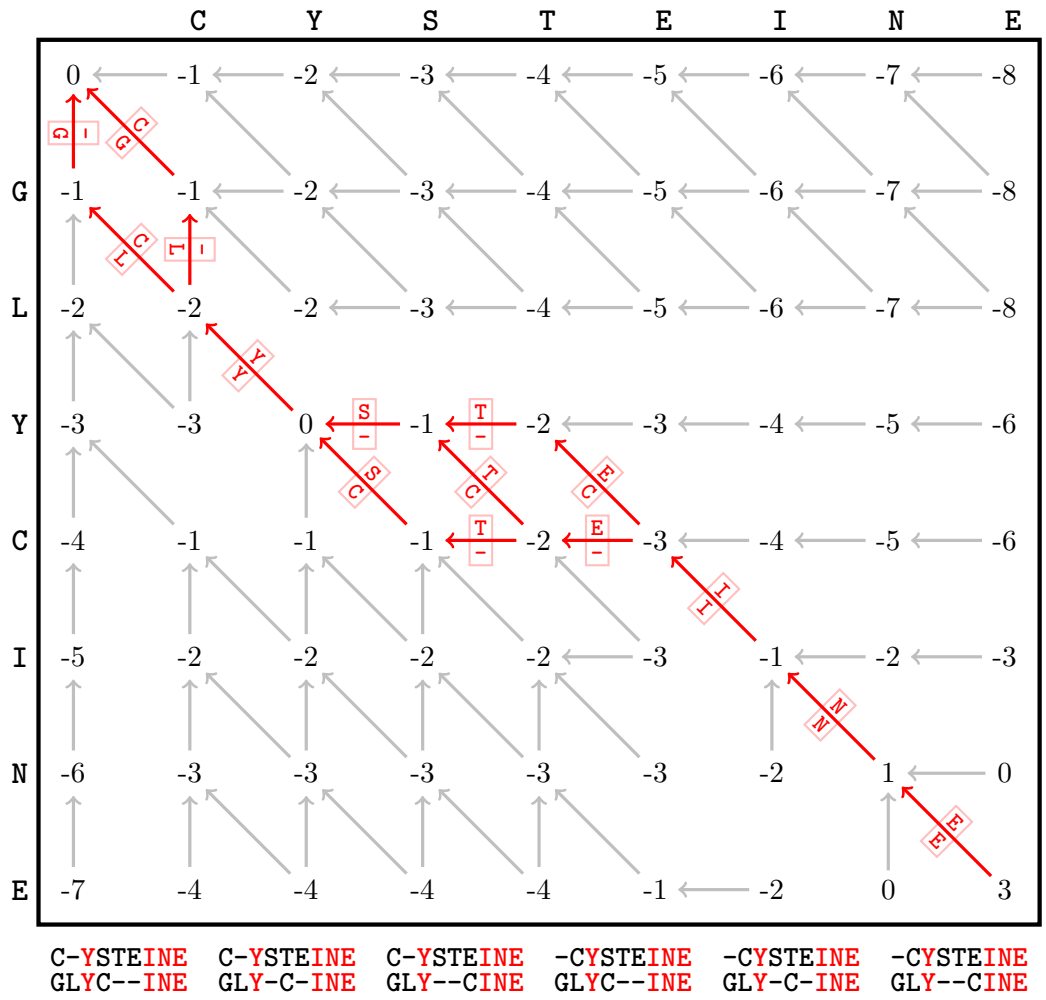

Figure 1.3.3: *Dynamic array filled with partial scores and backpointers, for the pairwise alignment of sequences* CYSTEINE *and* GLYCINE *with a match score of* 2*, a mismatch score of* -1*, and a gap penalty of* -1*. The best alignment score is* 3*, possible backtracking paths are drawn in red, and corresponding optimal alignments are showned under the array.*

Generalized Needleman-Wunsch-Gotoh

The choice of a constant gap penalty is not ideal. Looking at optimal alignments of figure 1.3.3, `C-YSTEINE` / `GLY-C-INE` and `-CYSTEINE` / `GLYC--INE` have the same score, but the latter alignment is better in a biological sense, because one gap of length two instead of two gaps of length one (in the bottom sequence) and one end gap instead of a gap between two residues (in the top sequence) are more biologically plausible. Therefore we need a scoring system which allows for variable gap penalties.

Scoring system. The one we will describe here use *affine* gap penalties and was first introduced by Gotoh [Got82] (hence we think that the algorithm should be more accurately named the Needleman-Wunsch-Gotoh algorithm, since most implementations use affine gaps). This means that a gap of length n will cost an affine penalty of $(d+n\cdot g)$

rather than a linear penalty of $(n \cdot g)$. The number d is called the *gap opening penalty* while the number g is the *gap extending penalty.*

Since the basic Needleman-Wunsch algorithm is going to be generalized in this section, we thought that we may as well kill two birds with one stone by allowing gap penalties to depend on their positions in the sequence, rather than on their lengths alone. This means an algorithm more general than the classical Needleman-Wunsch-Gotoh one, but since this generalization does not come with much additional complexity (and also because we were not able to find it in the current bioinformatics litterature), we decided to include it. Similarly, instead of having residue substitution scores $\mathrm{sub}(x_i, y_j)$, nothing prevents us from using more general position matching scores $\mathrm{sub}(i, j)$, something which will be useful later when using DynaMine data for aligning sequences.

In our formalism, the algorithm parameters are one $m \times n$ symmetric matrix noted 'sub' (with indices starting at 1) for substitution scores, and two $(m+1) \times (n+1)$ matrices noted 'gap_X' and 'gap_Y' (with indices starting at 0) for gap penalties in sequences $\mathbf{x}$ and $\mathbf{y}$ respectively. Usually, gap penalties are always nonpositive numbers; exceptions could occur if for example we believe that an insertion or deletion probably took place at a specific position. The numbers contained in these three matrices are defined precisely in figures 1.3.4 and 1.3.5.

$\mathrm{sub}(i,j)$	:=	score for matching x_i with y_j	
$\mathrm{gap}_X(i,0)$	:=	penalty for opening a gap between x_i and x_{i+1}	
$\mathrm{gap}_Y(0,j)$	:=	penalty for opening a gap between y_j and y_{j+1}	
$\mathrm{gap}_X(i,j)$	:=	penalty for matching a gap between x_i and x_{i+1} with y_j	$(j \neq 0)$
$\mathrm{gap}_Y(i,j)$	:=	penalty for matching a gap between y_j and y_{j+1} with x_i	$(i \neq 0)$

Figure 1.3.4: *Parameters for the generalized Needleman-Wunsch-Gotoh algorithm:* sub *is a* $m \times n$ *matrix with indices starting at* 1, gap_X *and* gap_Y *are* $(m+1) \times (n+1)$ *matrices with indices starting at* 0.

Recall that the two sequences were noted $\mathbf{x} := (x_1, \ldots, x_m)$ and $\mathbf{y} := (y_1, \ldots, y_n)$, so there are no residues noted x_0, x_{m+1}, y_0, or y_{n+1}. In the above figure, they are seen as 'virtual residues' used for defining the end gap penalties (e.g. a gap between x_0 and x_1 is a left end gap in the $\mathbf{x}$ sequence). How to set end gap penalties for each sequence is explained more clearly in figure 1.3.5.

left end gap opening penalty:	$\mathrm{gap}_X(0,0)$	and	$\mathrm{gap}_Y(0,0)$	
right end gap opening penalty:	$\mathrm{gap}_X(m,0)$	and	$\mathrm{gap}_Y(0,n)$	
left end gap extending penalties:	$\mathrm{gap}_X(0,j)$	and	$\mathrm{gap}_Y(i,0)$	$(i,j \neq 0)$
right end gap extending penalties:	$\mathrm{gap}_X(m,j)$	and	$\mathrm{gap}_Y(i,n)$	

Figure 1.3.5: *End gap parameters in the generalized Needleman-Wunsch-Gotoh algorithm.*

In order to score an alignment, it suffices again to compute the score of every column, and then to sum all the column scores. Besides the position-dependent scores and penalties, we now have to add a gap opening penalty to the score of each column containing a first gap. An example for the alignment $\begin{bmatrix} - & x_1 & x_2 & x_3 & x_4 & - & x_5 \\ y_1 & y_2 & - & - & y_3 & y_4 & y_5 \end{bmatrix}$ (written vertically) is shown in figure 1.3.6.

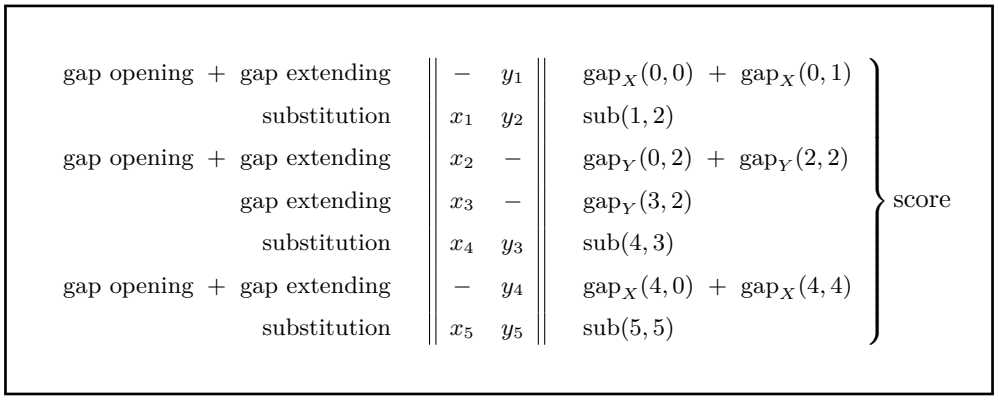

Figure 1.3.6: *Generalized Needleman-Wunsch-Gotoh alignment score calculation.*

Recursion relation. Now that the scoring system is defined, we need to find a recurrence relation for computing maximum subalignment scores. This is a bit more complicated this time, as we will use three dynamic arrays, each for a different kind of subalignment.

- $X(i,j)$:= max score of all (i,j)-subalignments ending with a gap in the **x**-subsequence: $\begin{bmatrix} \cdots x_i & - & - & - \\ \cdots & * & * & * \; y_j \end{bmatrix}$
- $Y(i,j)$:= max score of all (i,j)-subalignments ending with a gap in the **y**-subsequence: $\begin{bmatrix} \cdots & * & * & * \; x_i \\ \cdots y_j & - & - & - \end{bmatrix}$
- $Z(i,j)$:= max score of all (i,j)-subalignments ending with a matching of two symbols: $\begin{bmatrix} \cdots\cdots\cdots x_i \\ \cdots\cdots\cdots y_j \end{bmatrix}$

With a similar reasoning to the one used for deriving the basic Needleman-Wunsch recursion, we remark that each different kind of subalignment can always be built by appending a column to a smaller subalignment.

- $X(i,j)$: $\begin{bmatrix} \cdots & - \\ \cdots & y_j \end{bmatrix} = \left(\begin{bmatrix} \cdots & — \\ \cdots & y_{j-1} \end{bmatrix} \textbf{ or } \begin{bmatrix} \cdots & x_i \\ \cdots & — \end{bmatrix} \textbf{ or } \begin{bmatrix} \cdots & x_i \\ \cdots & y_{j-1} \end{bmatrix} \right) + \begin{bmatrix} - \\ y_j \end{bmatrix}$
- $Y(i,j)$: $\begin{bmatrix} \cdots & x_i \\ \cdots & - \end{bmatrix} = \left(\begin{bmatrix} \cdots & — \\ \cdots & y_j \end{bmatrix} \textbf{ or } \begin{bmatrix} \cdots & x_{i-1} \\ \cdots & — \end{bmatrix} \textbf{ or } \begin{bmatrix} \cdots & x_{i-1} \\ \cdots & y_j \end{bmatrix} \right) + \begin{bmatrix} x_i \\ - \end{bmatrix}$
- $Z(i,j)$: $\begin{bmatrix} \cdots & x_i \\ \cdots & y_j \end{bmatrix} = \left(\begin{bmatrix} \cdots & — \\ \cdots & y_{j-1} \end{bmatrix} \textbf{ or } \begin{bmatrix} \cdots & x_{i-1} \\ \cdots & — \end{bmatrix} \textbf{ or } \begin{bmatrix} \cdots & x_{i-1} \\ \cdots & y_{j-1} \end{bmatrix} \right) + \begin{bmatrix} x_i \\ y_j \end{bmatrix}$

Using these decompositions, we can now write recursion formulas for computing the partial scores in the X, Y, and Z dynamic arrays; see figure 1.3.7.

$$
\begin{aligned}
X(i,j) &= \mathrm{gap}_X(i,j) + \max\begin{cases} X(i\ ,\ j-1) & \\ Y(i\ ,\ j-1) & + \ \mathrm{gap}_X(i,0) \\ Z(i\ ,\ j-1) & + \ \mathrm{gap}_X(i,0) \end{cases} \\
Y(i,j) &= \mathrm{gap}_Y(i,j) + \max\begin{cases} X(i-1,\ j\ ) & + \ \mathrm{gap}_Y(0,j) \\ Y(i-1,\ j\ ) & \\ Z(i-1,\ j\ ) & + \ \mathrm{gap}_Y(0,j) \end{cases} \\
Z(i,j) &= \mathrm{sub}(i,j) + \max\begin{cases} X(i-1,\ j-1) \\ Y(i-1,\ j-1) \\ Z(i-1,\ j-1) \end{cases}
\end{aligned}
$$

Figure 1.3.7: *Recursion relation for the generalized Needleman-Wunsch-Gotoh algorithm.*

Of course, starting values must be defined. We again set $Z(0,0) := 0$ for the 'empty alignment', then for scores of nonexistent alignments (e.g. there is no $(1,0)$-subalignments ending with a gap in the **x**-sequence), we simply set scores of $-\infty$.

$$X(i,0) \ = \ Y(0,j) \ = \ Z(i+1,0) \ = \ Z(0,j+1) \ := -\infty \qquad \text{for all} \ \ i \geq 0 \text{ and } j \geq 0$$

Backtracking. When using the recursion relation for filling the three $(m+1)\times(n+1)$ matrices X, Y, and Z with partial scores, into each of their cells we should also store a pointer back to the cell from which the partial score was derived (there can be up to three different pointers per cell: store them all if you want to produce all possible optimal alignments). Then the backtracking part is easy: we start from the cell holding the best global score (so, among $X(m,n)$, $Y(m,n)$, and $Z(m,n)$, we pick the one containing the highest score), and then we just follow the pointers back to $Z(0,0)$ to build the alignment in reverse. Going to a X cell means adding a gap in the **x**-sequence, to a Y cell adding a gap in the **y**-sequence, and to a Z cell matching a x_i with a y_j.

It is difficult to provide a picture explaining a completed Needleman-Wunsch-Gotoh algorithm, because there are three arrays to depict. We attempted it in figure 1.3.8: it shows a $(m+1)\times(n+1)$ array of cells, with each (i,j) cell containing the values of $X(i,j)$, $Y(i,j)$, and $Z(i,j)$. The backtracking path is also drawn.

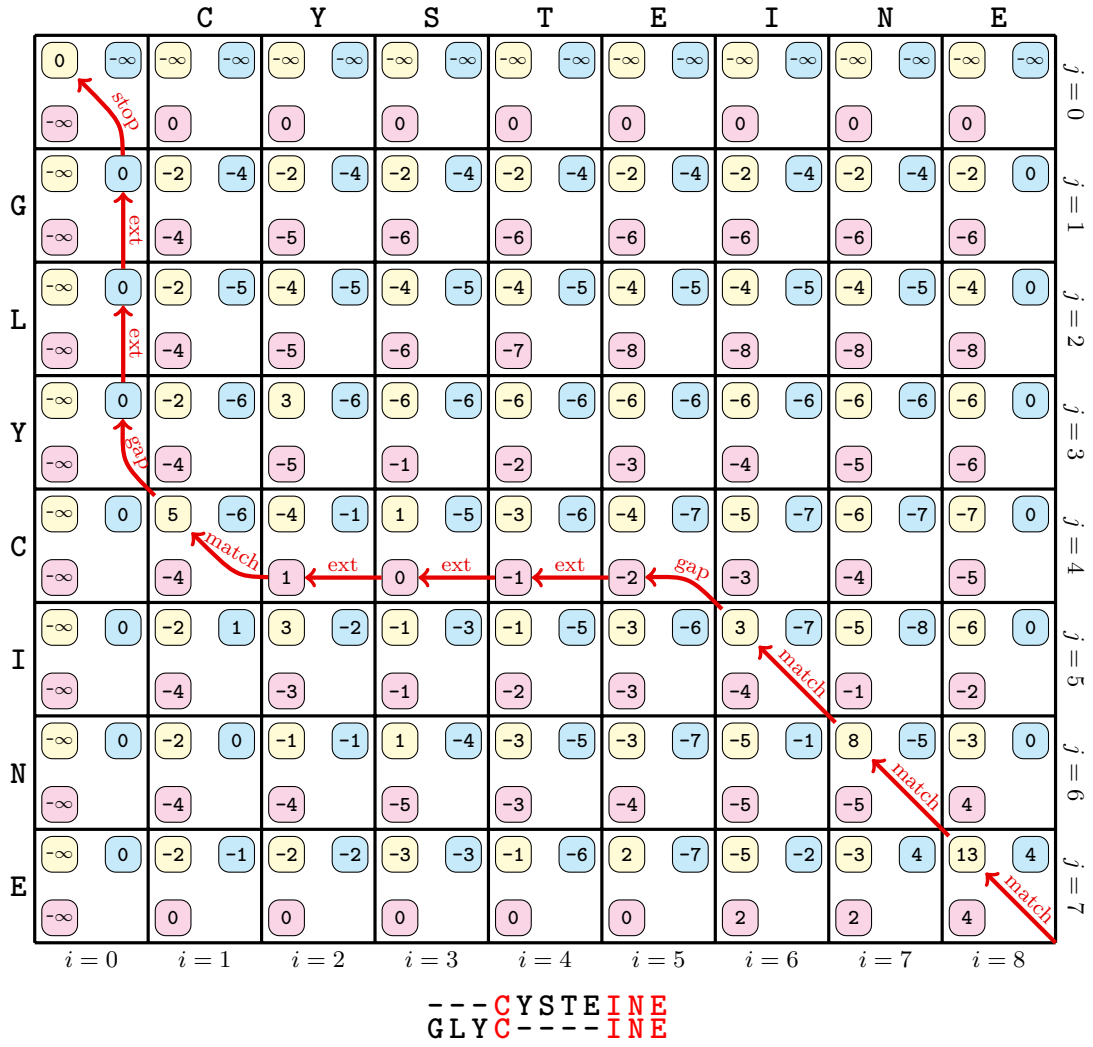

Figure 1.3.8: *The three dynamic arrays (X in **cyan**, Y in **magenta**, and Z in **yellow**) for the Needleman-Wunsch-Gotoh alignment of sequences* CYSTEINE *and* GLYCINE*, with a match score of* 5*, a mismatch score of* -2*, a gap opening penalty of* -3*, a gap extending penalty of* -1*, and no end gap penalties. The best alignment score is* 13*, and there is only one backtracking path (hence an unique optimal solution).*

Additional remarks on Needleman-Wunsch-Gotoh

Common parameters. In most implementations of the algorithm, such as the `needle` program of the EMBOSS software package [RLB00], gap penalties are not position-dependent. Rather, they let you choose a global gap opening penalty d and a global gap extending penalty g. In our notations, this means that $\mathrm{gap}_X(i,0) = \mathrm{gap}_Y(0,j) := d$ and $\mathrm{gap}_X(i,j) = \mathrm{gap}_Y(i,j) := g$, Also, substitution scores come from a scoring matrix, so $\mathrm{sub}(i,j) := M(x_i, y_j)$ where M is a BLOSUM matrix for example.

The advantage of our generalized algorithm is that it permits us to do things such as customizing gap penalties in certain parts of the protein, for example depending on the subsequence that will be inserted/deleted.

Consecutive indels. The Needleman-Wunsch-Gotoh algorithm as described in this section may allow a deletion to be directly followed by an insertion (and conversely), this means that it could produce alignments of the form $\begin{smallmatrix}\texttt{CYS--TE--INE}\\ \texttt{---GL--YCINE}\end{smallmatrix}$, something which may be seen as undesirable from a biologist's standpoint. But in practice such alignments almost never occur, and as we shall see, it is easy to derive conditions on the algorithm parameters which if satisfied will disallow consecutive indels in optimal alignments.

Suppose a (i,j)-subalignments is of the form $\begin{bmatrix}\cdots x_i & -\\ \cdots & y_j\end{bmatrix}$. If we replace its last two columns so that it becomes $\begin{bmatrix}\cdots x_i\\ \cdots y_j\end{bmatrix}$, we have added a residue substitution, removed an opening gap in the $\mathbf{x}$-sequence, and removed a gap in the $\mathbf{y}$-sequence. So the alignment score has changed by at least $\big(\mathrm{sub}(i,j) - \mathrm{gap}_X(i,0) - \mathrm{gap}_X(i,j) - \mathrm{gap}_Y(i,j-1)\big)$, and if we do not want the original subalignment to be optimal, this change should be positive. Therefore (sufficient) conditions for avoiding specific consecutive indels are:

$$\begin{bmatrix}\cdots x_i & - & \cdots\\ \cdots & y_j & \cdots\end{bmatrix} \text{ is not optimal } \quad \textbf{if} \quad \mathrm{gap}_X(i,0) + \mathrm{gap}_X(i,j) + \mathrm{gap}_Y(i,j-1) \;<\; \mathrm{sub}(i,j)$$

$$\begin{bmatrix}\cdots & x_i & \cdots\\ \cdots y_j & - & \cdots\end{bmatrix} \text{ is not optimal } \quad \textbf{if} \quad \mathrm{gap}_Y(0,j) + \mathrm{gap}_Y(i,j) + \mathrm{gap}_X(i-1,j) \;<\; \mathrm{sub}(i,j)$$

Or more simply, if d and g are the smallest opening and extending gap penalties respectively (without accounting end gaps if they are set to zero), and s is the lowest substitution score, then a sufficient condition for avoiding consecutive indels is $d + 2g < s$. In the case of global gap penalties, this condition is almost always satisfied: for example the default gap penalties of EMBOSS `needle` are $d = -10$ and $g = -0.5$, and the lowest mismatch score in the BLOSUM62 matrix is $s = -4$.

Computational complexity. The algorithm takes quadratic time in the size of the input: it is $\mathcal{O}(mn)$ with m and n being the lengths of the sequences to align. If we are just interested in the global score (and have no use for an actual optimal alignment), then it is possible to fill the arrays rows by rows (for example), discarding the rows previously computed. This allows for a linear space complexity, but then an optimal alignment can not be produced; only its score can.

However, Myers and Millers [MM88] were able to modify the algorithm so that it produces an optimal alignment in linear space, using a *divide and conquer* method common in the computer science literature [Hir75], but which at the time had still not be used in bioinformatics. As this algorithm is somewhat more complicated, we will not explain it here. In all cases, although it is possible to improve Needleman-Wunsch so that it has linear space complexity, the time complexity stays quadratic (because the dynamic arrays still need to be computed, even if we later discard some of its values).

Dynamic Time Warping

As a conclusion to the section, we will briefly describe another alignment algorithm, originally developed for matching continuous signals (in particular, time series) and used in a wide range of disciplines such as speech recognition [SC78] or biomedical

informatics [TGQS09]. Our motivation for including a short presentation of Dynamic Time Warping (DTW) in this thesis comes from the fact that it was actually one of the first method we tried for matching DynaMine data (although the approach was fruitless). We then discovered that the algorithm was a special case of the generalized Needleman-Wunsch-Gotoh (NWG) algorithm, and we think it could be interesting to explain how it is so. A good introduction can be found in [MÖ7], a book more concerned with analysis of music and audio data, but which provides a clear description of the DTW algorithm.

The DTW algorithm. From the point of view of DTW, two sequences $\mathbf{x} := (x_1, \ldots, x_m)$ and $\mathbf{y} := (y_1, \ldots, y_n)$ are seen as time series that need to be matched together, using local 'dilatations' so that the *distance* between them is minimized. Formally, a DTW alignment of $\mathbf{x}$ and $\mathbf{y}$ is a sequence of pairs of indices $(i_k,\, j_k)_{k=1\ldots\ell}$ satisfying:

$$\begin{array}{rcll} (i_1,\, j_1) = (1,1) & \text{and} & (i_\ell,\, j_\ell) = (m,n) & \textit{(Boundary condition)} \\ i_1 \leq \cdots \leq i_\ell & \text{and} & j_1 \leq \cdots \leq j_\ell & \textit{(Monotonicity condition)} \\ (i_{k+1},\, j_{k+1}) - (i_k,\, j_j) & \in & \{\, (0,1),\, (1,1)\, (1,0)\,\} & \textit{(Step size condition)} \end{array}$$

Taking again our examples of $\mathbf{x} =$ (CYSTEINE) and $\mathbf{y} =$ (GLYCINE), a possible DTW alignment would be $\left[\begin{smallmatrix} 1 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 1 & 2 & 3 & 3 & 3 & 4 & 5 & 6 & 7 \end{smallmatrix}\right]$, where each column is a pair of indices $(i_k,\, j_k)$. Using the sequence symbols rather than the indices, the alignment is $\left[\begin{smallmatrix} \texttt{C} & \texttt{C} & \texttt{Y} & \texttt{S} & \texttt{T} & \texttt{E} & \texttt{I} & \texttt{N} & \texttt{E} \\ \texttt{G} & \texttt{L} & \texttt{Y} & \texttt{Y} & \texttt{Y} & \texttt{C} & \texttt{I} & \texttt{N} & \texttt{E} \end{smallmatrix}\right]$. DTW align by repeating certain symbols, not by inserting gaps.

Once a *local distance* function $\mathrm{dist}(x, y)$ is given, we can compute the *total distance* $\sum_{k=1}^{\ell} \mathrm{dist}(x_{i_k},\, y_{j_k})$ of an alignment. For example, if the distance between two letters is defined as the absolute difference of their positions in the alphabet, then the total distance of $\left[\begin{smallmatrix} \texttt{C} & \texttt{C} & \texttt{Y} & \texttt{S} & \texttt{T} & \texttt{E} & \texttt{I} & \texttt{N} & \texttt{E} \\ \texttt{G} & \texttt{L} & \texttt{Y} & \texttt{Y} & \texttt{Y} & \texttt{C} & \texttt{I} & \texttt{N} & \texttt{E} \end{smallmatrix}\right]$ is $(4{+}9{+}0{+}6{+}5{+}2{+}0{+}0{+}0) = 26$. Of course, the goal of DTW is to produce an optimal alignment which minimizes the total distance.

As its names implies, the algorithm uses a dynamic programming method, and is very similar to Needleman-Wunsch. The minimum total distance $D(i, j)$ among all possible (i, j)-subalignments is defined, and the array is filled using a simple recursion formula:

$$D(i,j) \;=\; \mathrm{dist}(x_i,\, y_j) \;+\; \min \begin{cases} D(i \quad ,\; j-1) \\ D(i-1,\; j \quad) \\ D(i-1,\; j-1) \end{cases}$$

An optimal alignment (or, in DTW terminology, a *warping path*) can again be found by backtracking from $D(m, n)$ to $D(0, 0)$.

DTW with NWG. It should not be a surprise that DTW can be encoded into the much more general alignment algorithm that is Needleman-Wunsch-Gotoh. We just have to choose the right parameters, and then NWG will act like DTW. In the produced alignments, a gap will be understood as a repetition of the last encountered symbol, e.g.

$\begin{bmatrix}\texttt{C-YSTEINE}\\ \texttt{GLY--CINE}\end{bmatrix}$ is converted to $\begin{bmatrix}\texttt{CCYSTEINE}\\ \texttt{GLYYYCINE}\end{bmatrix}$. For a given distance function, NWG parameters must be set as shown below (indices i and j range from 1 to m and n respectively).

$$\begin{array}{rcl} & \text{sub}(i,j) := -\text{dist}(x_i,\, y_j) & \textit{(local distance)} \\ \text{gap}_X(i,j) = \text{gap}_Y(i,j) := -\text{dist}(x_i,\, y_j) & & \textit{(local distance)} \\ \text{gap}_X(i,0) = \text{gap}_Y(0,j) := 0 & & \textit{(no gap penalty)} \\ \text{gap}_X(0,0) = \text{gap}_Y(0,0) := -\infty & & \textit{(left boundary)} \end{array}$$

It is not difficult to understand how it works. We want to minimize a distance, but NWG maximizes a score, so on the first line we simply use negative distances as substitution scores. The second line means that gaps following a symbol are understood as repetitions of this symbol, e.g. a gap after x_i and matched to y_j is counted as a matching between x_i and y_j. Since DTW does not give a special penalty for repeating a symbol, gap opening penalties are disabled on the third line. Finally, the fourth line ensures that we have no gaps at the very beginning of the sequences, since there is no symbol before x_1 or y_1 to be repeated. Plugging all these parameters into the NWG recursion formulas (see figure 1.3.7), we obtain the following system:

$$\begin{array}{rcl} X(i,j) & = & -\text{dist}(x_i,\, y_j) + \max\{\, X(i\quad,j-1),\, Y(i\quad,j-1),\, Z(i\quad,j-1)\,\} \\ Y(i,j) & = & -\text{dist}(x_i,\, y_j) + \max\{\, X(i-1,j\quad),\, Y(i-1,j\quad),\, Z(i-1,j\quad)\,\} \\ Z(i,j) & = & -\text{dist}(x_i,\, y_j) + \max\{\, X(i-1,j-1),\, Y(i-1,j-1),\, Z(i-1,j-1)\,\} \end{array}$$

And recovering the DTW recursion from it is just a matter of three lines.

$$\begin{array}{rcl} D(i,j) & := & -\max\{\, X(i,j),\, Y(i,j),\, Z(i,j)\,\} \\ & = & \text{dist}(x_i,\, y_j) - \max\{\, -D(i,j-1),\, -D(i-1,j),\, -D(i-1,j-1)\,\} \\ & = & \text{dist}(x_i,\, y_j) + \min\{\, D(i,j-1),\, D(i-1,j),\, D(i-1,j-1)\,\} \end{array}$$

Therefore Dynamic Time Warping is a special case of Needleman-Wunsch-Gotoh.

1.4 Substitution and alignment scores

In the preceding chapter, the Needleman-Wunsch algorithm was described in details, but up to now, nothing has been said on how the parameters $\text{sub}(i,j)$, $\text{gap}_X(i,j)$ and $\text{gap}_Y(i,j)$ should be chosen. This is a very important matter: parameters should reflect our knowledge of sequence transformations, and may be different for aligning all proteins, or only proteins sharing a common characteristic or function (e.g. highly dissimilar proteins, or membrane proteins), or even non-protein sequences (e.g. DNA and RNA sequences, or sequences coming from outside biology). In this thesis, we shall use gap penalties that are the same everywhere in both sequences: so we will assume a constant opening penalty $\text{gap}_X(i,0) = \text{gap}_Y(0,j) := d$ and a constant extending penalty $\text{gap}_X(i,j) = \text{gap}_Y(i,j) := g$, with the exception of end gaps that will often be set to zero. Therefore we focus on substitution scores $\text{sub}(i,j)$, which in this section will only depend on the residues at positions i and j: $\text{sub}(i,j) := M(x_i, y_j)$ for some given amino

acid *substitution matrix* M. Of course the d and g gap penalties also have to be chosen, but this matter will not be discussed here; rather we will set them to the default values used by most aligner programs in bioinformatics. Let us just say that the gap penalties should be adjusted to the substitution matrix (or conversely), for example having larger values in M should come with larger gap penalties.

It should be stressed out that an *alignment score* can mean two different things: the number determined by the choice of gap penalties and substitution scores (this is the score which the Needleman-Wunsch algorithm maximizes), but also the quality of a given alignment compared to a reference alignment (believed to be correct) of the same sequences (this way, we can know if our choice of parameters is good). Which score we will discuss should be clear from the context.

We begin the section with a short overview of substitution matrices, including the description of a 'naive' matrix which, although never used in modern bioinformatics, is very simple to derive while still carrying some biological meaning. Then the BLOSUM family of matrices will be described in details, and finally we will explain how to score an alignment quality when comparing it to a reference alignment.

Amino acid substitution matrices

Generally, an amino acid substitution matrix is a 20×20 symmetric matrix M of numbers, containing scores $M(x, y)$ for each $x \leftrightarrow y$ substitution of amino acids x and y. These scores should be *additive*: it means that we may add them together for computing a score for several substitutions occuring simultaneously in an alignment (which is how the Needleman-Wunsch scoring system works). In particular, scores are not probabilities, which would be *multiplicative* rather than additive. However, the construction of substitution matrices often begin by computing probabilities, before converting them to additive scores.

A basic example using biological knowledge. Besides the trivial case of using a match score and a mismatch score, for example $M(x, y) = \begin{cases} 1 \text{ if } x{=}y \\ 0 \text{ if } x{\neq}y \end{cases}$, the oldest example of an amino acid substitution matrix we could find is described in the original Needleman and Wunsch article [NW70]. After describing their algorithm, the authors use it with a matrix derived from the DNA codon table. Recall that a codon is a triplet of nucleotides encoding a specific amino acid (different codons can translate to the same residue), and the set of codon-residue translation rules (the genetic code) is traditionally represented in a DNA (or RNA) codon table. The authors' idea was to set for each pair of amino acids the maximum number of corresponding bases in their respective codons. For example, `M` (methionine) is encoded by a `ATG` codon and `Q` (glutamine) can be encoded by both `CAA` and `CAG` codons. There is no corresponding nucleotides in $\left[\begin{smallmatrix}\texttt{ATG}\\\texttt{CAA}\end{smallmatrix}\right]$ but there is one in $\left[\begin{smallmatrix}\texttt{ATG}\\\texttt{CAG}\end{smallmatrix}\right]$, so the score for a `M`$\leftrightarrow$`Q` substitution is set to 1. This method gives rise to a substitution matrix with scores in $\{0, 1, 2, 3\}$, depicted in figure 1.4.1. In the article the two authors try their algorithm with different (linear) gap penalties and variations

of this matrix (replacing $\{0, 1, 2, 3\}$ by other values).

Corresponding DNA codons		A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
GCT GCC GCA GCG	**A**	3	1	1	2	1	1	2	2	1	1	1	1	1	1	2	2	2	1	1	2
CGT CGC CGA CGG AGA AGG	**R**	1	3	1	1	2	2	1	2	2	2	2	2	2	1	2	2	2	2	1	1
AAT AAC	**N**	1	1	3	2	1	1	1	1	2	2	1	2	1	1	1	2	2	0	2	1
GAT GAC	**D**	2	1	2	3	1	1	2	2	2	1	1	1	0	1	1	1	1	0	2	2
TGT TGC	**C**	1	2	1	1	3	0	0	2	1	1	1	0	0	2	1	2	1	2	2	1
CAA CAG	**Q**	1	2	1	1	0	3	2	1	2	1	2	2	1	0	2	1	1	1	1	1
GAA GAG	**E**	2	1	1	2	0	2	3	2	1	1	1	2	1	0	1	1	1	1	1	2
GGT GGC GGA GGG	**G**	2	2	1	2	2	1	2	3	1	1	1	1	1	1	1	2	1	2	1	2
CAT CAC	**H**	1	2	2	2	1	2	1	1	3	1	2	1	0	1	2	1	1	0	2	1
ATT ATC ATA	**I**	1	2	2	1	1	1	1	1	1	3	2	2	2	2	1	2	2	0	1	2
TTA TTG CTT CTC CTA CTG	**L**	1	2	1	1	1	2	1	1	2	2	3	1	2	2	2	2	1	2	1	2
AAA AAG	**K**	1	2	2	1	0	2	2	1	1	2	1	3	2	0	1	1	2	1	1	1
ATG	**M**	1	2	1	0	0	1	1	1	0	2	2	2	3	1	1	1	2	1	0	2
TTT TTC	**F**	1	1	1	1	2	0	0	1	1	2	2	0	1	3	1	2	1	1	2	2
CCT CCC CCA CCG	**P**	2	2	1	1	1	2	1	1	2	1	2	1	1	1	3	2	2	1	1	1
TCT TCC TCA TCG AGT AGC	**S**	2	2	2	1	2	1	1	2	1	2	2	1	1	2	2	3	2	2	2	1
ACT ACC ACA ACG	**T**	2	2	2	1	1	1	1	1	1	2	1	2	2	1	2	2	3	1	1	1
TGG	**W**	1	2	0	0	2	1	1	2	0	0	2	1	1	1	1	2	1	3	1	1
TAT TAC	**Y**	1	1	2	2	2	1	1	1	2	1	1	1	0	2	1	2	1	1	3	1
GTT GTC GTA GTG	**V**	2	1	1	2	1	1	2	2	1	2	2	1	2	2	1	1	1	1	1	3

Figure 1.4.1: *A 'naive' substitution matrix derived from the DNA codon table alone.*

If we gave this example, it is only because it is easy to understand, built using biological knowledge, and is historically one of the earliest amino acid substitution matrix. But is not a 'good' matrix, and to our knowledge matrices built with this simplistic method are not used in real modern bioinformatics; in fact Needleman and Wunsch were not trying to derive good scores for amino acid substitutions, they just wanted something to try their new algorithm. One obvious problem with this method is that it assumes that the only possible mutations in a genome are nucleotide substitutions or indels of a number of nucleotides divisible by three. But a single nucleotide insertion or deletion could also happen, changing the subsequent grouping of the codons and thus resulting in a completely different translation of the rest of the sequence (such a phenomenon is called a *frameshift mutation*). For example, (GATCCGTGCATT$\cdots$) translates to (**DPCI**$\cdots$), but (GAATCCGTGCATT$\cdots$) translates to (**ESVH**$\cdots$). Moreover, building a matrix from a codon table alone completely ignores the evolutionary mechanisms responsible for a protein's existence in the first place. Indeed, a substitution can modify the structure or function of a protein (e.g. when an amino acid which is hydrophobic is replaced by one which is not), in which case the modified protein may be rejected by the processes of natural selection (e.g. because it prevents its host organism from reproducing by rendering it infertile); therefore that particular substitution would be less likely to occur.

Inferring scores using biological data. The first successful substitution matrices in bioinformatics are probably the PAM matrices, introduced by Dayhoff in 1978 [DS78]. This family of matrices was calculated from observed mutations in the phylogenetic trees of 71 families of closely related proteins: hence the substitution score are inferred from known biological data (in this case, an evolutionary history of proteins). The PAM name

comes from *point accepted mutation*, which is an amino acid substitution accepted by natural selection; and the probability of a substitution to be accepted can be estimated from a phylogenetic tree. These substitution probabilities are then converted to a matrix of substitution scores that can be used in the Needleman-Wunsch algorithm.

This approach, using reference biological data, is the most common way of creating substitution matrices; of course, the question of *how* to generate a reference dataset in the first place still has to be answered. One possible method is to use a set of structural alignments of proteins whose 3D structures were experimentally resolved [PDS00]. Since structure is more conserved than sequence [IAE09], structural alignments are probably of good quality, hence we can use them to learn how to align sequences with unknown 3D structures. The reference dataset may also be chosen depending on which kind of proteins we want to align, for example substitution matrices for aligning transmembrane proteins have been created in this manner [NHH00].

The BLOSUM substitution matrices [HH92], which are maybe the ones most commonly used in bioinformatics today, were also inferred from reference biological data. In this thesis we are especially interested in this family of matrices: in fact custom BLOSUM matrices will be generated in the later chapters. For this reason, the algorithm for creating them will now be described in details.

BLOSUM matrices

The procedure for creating BLOSUM matrices, first described by Henikoff and Henikoff in 1992 [HH92] (a more modern and informal presentation can also be found in [Edd04]), can be summarized in the following steps:

1) Choose a reference dataset of gap-free alignments (called *blocks*).
2) Cluster similar sequences in each block.
3) Compute observed and expected substitution probabilities.
4) Compute substitution likelihood ratios ($\frac{\text{observed probabilities}}{\text{expected probabilities}}$).
5) Substitution scores are logarithms of likelihoods ratios.

A block is a multiple sequence alignment devoid of any gap (hence all sequences in a block have the same length). If we want to generate BLOSUM matrices from a dataset of gapped alignments, we need to first convert it to a dataset of blocks. For example, each alignment could either be stripped of its gaps-containing columns, or be splitted into several blocks of a minimum size. The second point is important: the clustering threshold may be chosen, so that different BLOSUM matrices may be generated from a same dataset. If we decide to cluster together sequences with more than T% residue identity, the resulting matrix is called a BLOSUMT matrix. Higher clustering thresholds will produce matrices designed for aligning more closely related sequences. The remaining steps are just the calculations of what are usually called log-odd scores in bioinformatics, or log-likelihood ratios in statistics.

We will begin with the formulas for computing the numbers in the last three steps, assuming no clustering at all (i.e. $T = 100\%$). The second step (clustering) is a bit more

complicated, but once it is done the formulas of the remaining steps stay unchanged, hence we prefer to explain the clustering part afterwards. Also, in what follows, a 'residue' is not necessarily an amino acid but just a symbol taken from an arbitrary alphabet: we want our description to be general, so that it can later be used for any kind of sequences (of course, what we really have in mind are sequences of DynaMine values). Without loss of generality, we will in fact take the $\{1, 2, 3, \ldots, R\}$ set of numbers as our residue alphabet (so in the case of proteins we just have $R = 20$ with residue x being the xth amino acid).

Computing scores (without clustering). In order to compute substitution probabilities, we have to count, for each pair (x, y) of residues, the number of $x \leftrightarrow y$ substitutions in the dataset. More precisely, we first define a $R \times R$ substitution frequency array $F(x, y)$ that is initialized with zero everywhere. Then for each block in the dataset, we loop on each possible pair $(\mathbf{s}, \mathbf{t})$ of sequences coming from this block, count the number of $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ columns in the $\left[\begin{smallmatrix} s_1 & \cdots & s_N \\ t_1 & \cdots & t_N \end{smallmatrix}\right]$ pairwise alignment, and add that number to $F(x, y)$. Equivalently, we can increment $F(s_n, t_n)$ for each column $\left[\begin{smallmatrix} s_n \\ t_n \end{smallmatrix}\right]$ in the pairwise alignment, with n going from 1 to N (the number of columns). By looping on pairs of sequences coming from a block, we mean ordered pairs: both $(\mathbf{s}, \mathbf{t})$ and $(\mathbf{t}, \mathbf{s})$ pairs, with $\mathbf{s} \neq \mathbf{t}$, have to be considered. So if there are M sequences in a block, we have to count the number of $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ columns in $(M^2 - M)$ pairwise alignments.

If done correctly, the resulting substitution frequency array $F(x, y)$ should be symmetric with even numbers on its diagonal. The substitution scores are then computed using the formulas in figure 1.4.2 (all matrices defined by these formulas are of course symmetric). Remark that our counting method differs from the one presented in [HH92], resulting in a frequency array different from the one in the original article. But the formulas below were adapted so that in the end the same scores are computed. If we chose to do things a bit differently, it is simply because we think it allows for prettier formulas

$$
\begin{aligned}
\mathrm{obs}(x, y) &:= F(x, y) \;/\; \sum_{i=1}^{R} \sum_{j=1}^{R} F(i, j) && \textit{(observed probabilities)} \\
\mathrm{exp}(x, y) &:= \Big(\sum_{j=1}^{R} \mathrm{obs}(x, j) \Big) \cdot \Big(\sum_{i=1}^{R} \mathrm{obs}(i, y) \Big) && \textit{(expected probabilities)} \\
\mathrm{rat}(x, y) &:= \mathrm{obs}(x, y) \;/\; \mathrm{exp}(x, y) && \textit{(likelihood ratios)} \\
\mathrm{score}(x, y) &:= \frac{1}{\lambda} \log \Big(\mathrm{rat}(x, y) \Big) && \textit{(substitution scores)}
\end{aligned}
$$

Figure 1.4.2: *Formulas for computing BLOSUM scores from a frequency array $F(x, y)$*

It is easy to make sense of these formulas. If we have a $\left[\begin{smallmatrix} x \\ y \end{smallmatrix}\right]$ column in a pairwise alignment, according to our reference dataset the probability of it happening because of a $x \leftrightarrow y$ substitution is $\text{obs}(x, y)$ while the probability of it happening by chance is $\exp(x, y)$. Therefore the likelihood ratio $\text{rat}(x, y)$ expresses how many times the substitution hypothesis is more likely than the by-chance hypothesis. We then conveniently assume that aligned pairs are independent of each other (although it is biologically unlikely), allowing us to compute a global likelihood ratio for the pairwise alignment by multiplying the individual ratios of each aligned pair. However, we want *additive* substitution scores, that can be summed to get a global substitution score. For this reason we use a logarithm of the likelihood ratio, turning multiplication into addition. The λ constant is just a number that lets us scale the scores so that they can be rounded to nice integers.

Clustered frequencies. A problem that can arises when the dataset contains highly similar sequences that are not clustered together (i.e. $T = 100\%$), is that too many $x \leftrightarrow y$ substitutions will be counted in the $F(x, y)$ frequency array; something which is undesirable if we plan to use the generated matrix for aligning sequences with low similarity. The solution used in [HH92] is to cluster together the similar sequences appearing in each dataset block, assigning a weight of 1 to each cluster.

Clustering is a task that can be realized using different methods (see [XW$^+$05] and [Ber06] for surveys of clustering algorithms), and the chosen one often depends on the structure of the data to be clustered (for example, its dimensionality). Different algorithms will yield different clusters of sequences, hence different BLOSUM matrices. In their [HH92] article, Henikoff and Henikoff do not name the exact clustering algorithm they use: instead they describe it using an example, that is reproduced below using their own words:

> *For example, if the percentage is set at 80%, and sequence segment A is identical to sequence segment B at* $\geq$80% *of their aligned positions, then A and B are clustered and their contributions are averaged in calculating pair frequencies. If C is identical to either A or B at* $\geq$80% *of aligned positions, it is also clustered with them and the contributions of A, B, and C are averaged, even though C might not be identical to both A and B at* $\geq$80% *of aligned positions.*

If we understood that extract correctly, their method is what is usually called *single-linkage agglomerative hierarchical clustering*. The 'agglomerative hierarchical' part means that the algorithm starts with each element in a cluster of its own, which are then iteratively merged together to obtain larger clusters [Joh67]. Then 'single-linkage' means that the similarity between two clusters is the largest similarity between their elements, i.e. the similarity between a *single* pair of elements: namely the two (one in each cluster) that are the most similar. Two clusters may then be merged together if their similarity is at least $T\%$. In mathematical terms, if sequence similarity is noted $\text{sim}(\mathbf{s}, \mathbf{t})$:

$$\text{clusters } A \text{ and } B \text{ may be merged} \iff \max_{(\mathbf{s},\mathbf{t}) \in A \times B} \Big(\text{sim}(\mathbf{s}, \mathbf{t}) \Big) \geq T$$

Clusters are then iteratively merged until all remaining pairs of clusters have a similarity less than $T\%$. The order in which we merge the clusters is irrelevant: it can be proven

that we will always end with the same set of clusters. This clustering algorithm for one block is described using pseudocode in figure 1.4.5.

Input: a block and a clustering threshold T

Output: a set of clustered sequences $\mathscr{C}$

- $\mathscr{C}$:= $\{$ $\{\mathbf{s}\}$ for all sequences $\mathbf{s}$ in the block $\}$
- **loop** on :
  - **pick** distinct clusters $A, B \in \mathscr{C}$ with $\max_{(\mathbf{s},\mathbf{t}) \in A \times B} \Big(\mathrm{sim}(\mathbf{s},\mathbf{t}) \Big) \geq T$
  - **if** these two clusters exist :
    - $\mathscr{C}$:= $\{$ $C \in \mathscr{C}$ with $C \neq A$ and $C \neq B$ $\}$ $\cup$ $\{A \cup B\}$
  - **else** :
    - **return** $\mathscr{C}$ (quitting the loop)

Figure 1.4.3: *Clustering algorithm for a block.*

Once the clustering is done, we assign a $1/c$ weight to each sequence belonging to a cluster of size c, so that a whole cluster has a weight of 1 and two sequences belonging to the same cluster are not compared together. The resulting $F(x, y)$ frequencies are called *clustered frequencies*.

clustered sequences when $T = 60\%$

max similarity with sequences outside the cluster		*max similarity with other sequences in the cluster*
38%	TRDVDCDNIMSTNLFHCKDKNTFIYSRPEPVKAICKGIIASKNVLTTSEF	*N/A*
56%	DRYCERMMKRRSLTSPCKDVNTFIHGNKSNIKAICGANGSPYRENLRMSK	*N/A*
40%	TYCNQMMQRRGMTSPVCKFTNTFVHASAASITTVCGSGGTPASGDLRDSN	*N/A*
46%	LQCNKAMSGVNNYTQHCKPENTFLHNVFQDVTAVCDMPNIICKNGRHNCH	*N/A*
54%	SYCNLMMQRRKMTSHQCKRFNTFIHEDLWNIRSICSTTNIQCKNGQMNCH	92%
52%	AYCNLMMQRRKMTSHYCKRFNTFIHEDIWNIRSICSTSNIQCKNGQMNCH	92%
52%	NYCNLMMKARDMTSGRCKPLNTFIHEPKSVVDAVCHQENVTCKNGRTNCY	70%
52%	NYCNEMMKKREMTKDRCKPVNTFVHEPLAEVQAVCSQRNVSCKNGQTNCY	80%
54%	NYCNQMMMRRKMTQGRCKPVNTFVHESLEDVKAVCSQKNVLCKNGRTNCY	86%
52%	NYCNVMMIRRNMTQGRCKPVNTFVHESLADVQAVCFQKNVLCKNGQTNCY	88%
50%	NYCNQMMQSRNLTQDRCKPVNTFVHESLADVQAVCFQKNVACKNGQSNCY	92%
50%	NYCNQMMKSRNLTQSRCKPVNTFVHESLADVQAVCSQKNVACKNGQTNCY	98%
50%	NYCNQMMKSRNLTQGRCKPVNTFVHESLADVQAVCSQKNVACKNGQTNCY	98%
40%	QQCTNAMQVINNYQRRCKNQNTFLLTTFANVVNVCGNPNMTCPSNKTRKN	68%
38%	PRCTIAMRAINNYRWRCKNQNTFLRTTFANVVNVCGNQSIRCPHNRTLNN	68%
56%	DRYCESIMRRRGLTSPCKDINTFIHGNKRSIKAICENKNGNPHRENLRIS	66%
52%	DEYCFNMMKNRRLTRPCKDRNTFIHGNKNDIKAICEDRNGQPYRGDLRIS	66%
36%	NCNTIMDNNIYIVGGQCKRVNTFIISSATTVKAICTGVINMNVLSTTRFQ	66%
34%	DCNTIMDKAIYIVGGKCKERNTFIISSEDNVKAICSGVSPDRKELSTTSF	76%
38%	NCNTIMDKSIYIVGGQCKERNTFIISSATTVKAICSGASTNRNVLSTTRF	76%

Figure 1.4.4: *Clustering of similar sequences inside a block.*

An example of a block with clustered sequences is shown in figure 1.4.4. Remark that instead of using clustered frequencies, some authors prefer to replace every cluster by a consensus sequence; this is simpler than weighting sequences, however this not the method described in [HH92].

The exact algorithm for generating the $F(x, y)$ is described in figure 1.4.5 ; if we set $T := 100\%$, this is the same method as described earlier. In this algorithm, we suppose that each sequence in a block is numbered from 1 to M (the number of sequences in the block). Once the $F(x, y)$ array is computed, substitution scores for a BLOSUMT matrix can be derived from the same formulas than in figure 1.4.2.

Input: a set of blocks and a clustering threshold T

Output: clustered frequencies $F(x, y)$

- $F(x, y) := 0$ for all pairs (x, y) of residues
- **for** each block in the dataset :
  - $M :=$ number of sequences in the block
  - $N :=$ length of sequences in the block
  - **cluster** together sequences with more than $T\%$ residue identity
  - **for** each (s, t) with $1 \leq s < t \leq M$:
    - **if** sequences s and t belong to different clusters :
      - $u :=$ number of sequences in the cluster containing sequence s
      - $v :=$ number of sequences in the cluster containing sequence t
      - **for** each n with $1 \leq n \leq N$:
        - $x :=$ residue at position n in sequence s
        - $y :=$ residue at position n in sequence t
        - $F(x, y) := F(x, y) + 1 / (u \cdot v)$
        - $F(y, x) := F(y, x) + 1 / (v \cdot u)$
- **return** $F(x, y)$

Figure 1.4.5: *Algorithm for computing clustered frequencies from a set of blocks.*

Measuring alignment quality

Suppose that we just computed a pairwise alignment using some algorithm: how 'good' is this alignment? The most common way to answer this question is to compare the computed alignment to a reference alignment (of the same two sequences) which is believed to be 'correct'; for example because it is a structural alignment, or because it comes from a known phylogenetic tree of proteins. A score measuring how close the computed alignment is to the reference alignment can then be computed using different methods (but in this thesis we will focus on only one of these). And if a benchmark database of reference alignments is given, quality assessment of different

alignment algorithms becomes possible [LS05] [LS02] [Elo02]. Remark that in the rest of this section, an 'alignment score' means a measure of alignment quality, i.e. a number which is large when the considered alignment is close to its corresponding reference alignment. It should not be confused with the number that the Needleman-Wunsch algorithm wants to maximize.

The alignment score we chose to use is the *sum-of-pairs score*. Its idea is very simple: we just take the percentage of aligned pairs in the computed alignment that are also in the reference alignment; an example is provided in figure 1.4.6. For the moment we only consider sum-of-pairs scores for pairwise alignments, but in section 3.2 an extension to multiple alignments will be described. Because of the simplicity of the sum-of-pairs scoring method, it is difficult to find out where and when it was originally defined, but the method is described in most articles concerned with benchmark databases and alignment quality assessment, such as [TPP99b], or the three ones cited earlier.

Number of pairs in the reference pairwise alignment: **52**

```
FKIIASQCTSCSACEPLCPNVAI-SEKGGNFVI---EAA-KCSECVGHFDEPQCAAACPVDNTCVVDR
|||||||||||||||||||||| |||||||||    ||| ||||||||          ||   |||||||
VQIDEAKCIGCDTCSQYCPTAAIFGEMGEPHSIPHIEACINCGQCLTH---------CP--ENAIYEA
```

Number of pairs in the computed pairwise alignment,
that are also present in the reference pairwise alignment: **34**

```
FKIIASQCTSCSACEPLCPNVAISEKGG-----NFVIEAAKCSECVGHFDEPQCAAACPVDNTCVVDR
||||||||||||||||||||                ||                ||   |||||||
VQIDEAKCIGCDTCSQYCPTAAIFGEMGEPHSIPHIEACINC---------GQCLTHCP--ENAIYEA
```

Sum-of-pairs score of the computed pairwise alignment,
with respect to the reference pairwise alignment: **34/52 = 65%**

Figure 1.4.6: *Sum-of-pairs score for a computed pairwise alignment, with respect to a reference pairwise alignment of the same couple of sequences.*

Much criticism could be made, and has been made [Edg10] [JB09], on using the sum-of-pairs method for measuring alignment quality. A basic example where the method may give an unsatisfactory score is :

```
-VTG---EINPTRAPDIRGPVSLAF        --VTG---EINPTRAPDIRGPVSLAF
ESDRLALNDVR----RIRGPIS---        ESDRLALNDVR----RIRGPIS----
```

(reference alignment) *(computed alignment)*

Both reference and computed alignments are very similar: their internal gaps are inserted at the same place. However, the top sequence in the computed alignment is translated by one residue to the right, because of one additional opening gap. This cause all pairs to be aligned differently than in the reference, yielding a sum-of-pairs score of 0%, although visually the two alignments are almost the same.

Chapter 2

Design of the experiments

2.1 Outline of the experiments

We would like to use this short section to explain in more details what will actually be carried out in this thesis, and which data and tools we shall be using. Our general motivation is to align sequences using the Needleman-Wunsch-Gotoh algorithm described in section 1.3, but by incorporating DynaMine-predicted data into the algorithm parameters. There are many ways to do this, but we will use the following procedure:

1) Choose a benchmark database containing datasets of references multiple alignments
2) Truncate every sequence so that they are devoid of end gaps
3) Run DynaMine on every sequence in the dataset
4) Put DynaMine values in $[0, 1]$ into 50 equal-width bins
5) Partition the data into a training set (for inferring matrices) and a test set (for aligning)
6) Recreate the classical seqBLOSUM matrices, to ensure the implementation correctness
7) Generate 50×50 dynBLOSUM matrices for scoring matchings of DynaMine bins
8) Normalize dynBLOSUM matrices to avoid undesirable expected scores
9) Implement an aligner program that can use both seqBLOSUM and dynBLOSUM
10) Extend the sum-of-pairs score to multiple alignments and whole datasets
11) Find out the best seqBLOSUM using our aligner and scoring system
12) Use our aligner with different weighted averages of seqBLOSUM and dynBLOSUM
13) Investigate the averaging method improvements in the case of dissimilar sequences

This procedure is just a quick summary of what will be done in the coming sections. The reasoning behind each step will be made more precise, and we will of course discuss the obtained results. Also, by seqBLOSUM we mean a 20×20 BLOSUM matrix containing scores for amino acid substitutions, and by dynBLOSUM a 50×50 BLOSUM matrix containing scores for DynaMine values matchings. This terminology will be used throughout this chapter and the next.

Tools and data used

Our main scripting language is Python [VRD11], often used with the SciPy/NumPy [Bre12] and BioPython [`biopython.org`] libraries, and of course we used the MatPlotLib [`matplotlib.org`] library for generating most of our plots. Python was used for most small tasks, such as parsing text files, but also for implementing the BLOSUM generating algorithm. The language used for implementing the Needleman-Wunsch-Gotoh algorithm is the C programming language [KR88]. Besides programming languages, we also used some programs from the EMBOSS software package [RLB00], namely `seqret` and `needle`. The operating systems on which computing was done were either Arch Linux [`archlinux.org`] (for small tasks), or Scientific Linux [`scientificlinux.org`] (for more computationally expensive tasks: this OS was the one installed on a computer cluster at the university that I was allowed to use); of course, it came with extensive usage of the Bash Unix shell [RF02].

The data was taken from the BAliBASE [TPP99a,TPP99b,TKRP05] benchmark database; although it suffers from some criticism [Edg10,JB09] (but this is the case of any benchmark database), this is one of the most widely used, and also the one suggested by my thesis advisor. Other possible choices would have been SABmark [VWLW05] (which was coincidentally developed at the *Vrije Universiteit Brussel*), or one of those discussed in the [BWLH06] survey article.

2.2 Running DynaMine on the BAliBASE database

The last version of the BAliBASE database was obtained from the website of the *LBGI Bioinformatique et Génomique Intégratives* research group at the *Université de Strasbourg*: `http://lbgi.fr/balibase/`. It comes in a compressed archive containing multiple sequence alignments in the MSF, RSF, and XML file formats, and the alignments are grouped in 6 different subsets (described on figure 2.2.1). Files are named following the pattern `BBxxyyy`, with `xx` being the two-digit dataset identifier and `yyy` the three-digit sequence identifier. Similarly, files for the truncated alignments (see next paragraph) are named in the `BBSxxyyy` pattern.

dataset	description
RV11	equi-distant sequences with <20% identity
RV12	equi-distant sequences with 20–40% identity
RV20	families aligned with a highly divergent "orphan" sequence
RV30	subgroups with <25% residue identity between groups
RV40	sequences with N/C-terminal extensions
RV50	internal insertions

Figure 2.2.1: *The 6 different datasets in the BAliBASE database.*

With the exception of RV40, each dataset is further subdivided into two subsets: one

containing complete alignments, and one with the same alignments truncated so that all end gaps are eliminated and only the core block of alignments remain (see figure 2.2.2 for an example). The RV40 dataset is an exception, and only exist in a complete-alignments version, because its purpose is to test the effect of long terminal extensions, i.e. its alignments all contain a few sequences much longer than the other ones, hence long end gaps are required in any good sequence alignment. Therefore it would not make much sense to truncate the RV40 alignments.

```
---GKGDP  KKPRGKSYAFFVQTSREEHKKKHPDASVNFSEFSKKCSERWKT----EEDAKADKARYEREMKTYIPPKGE  ----------
------MQ  DRVKRPNFIVWSRDQRRKMALENP--RMRNSEISKQLGYQWKMLTEAEFQAQKLQAMHREKYPNYKYRPRR  KAKMLPK---
MKKLKKHP  DFPKKPTYFRFFMEKRAKYAKLHP--EMSNLDLTKILSKKYKELPEKKIQFQREKQEFERNLARFREDHPD  LIQNAKK---
--------  MHIKKPNFMLYMKEMRANVVAEST--LKESAAINQILGRRWHALSREEYEARK----HMQLYPGWSARDNY  GKKKKRKREK
```

Figure 2.2.2: *In their truncated versions, alignments have been stripped of their left and right end gaps, keeping only the center block (in bold in this example).*

We chose to conduct the test on the truncated alignments only, thus ignoring the RV40 dataset. Statistics on the size of the relevant datasets are summarized in the table on figure 2.2.3.

dataset	RV11	RV12	RV20	RV30	RV50	total
# alignments	38	44	41	30	16	169
# sequences	261	396	1 869	1 895	447	4 868
# residues	66 304	119 542	470 228	510 425	154 986	1 321 485

Figure 2.2.3: *Sizes of the datasets used in the experiments.*

DynaMine had to be run on each sequence in the database, so that we can use the resulting data for experimenting with alignments. DynaMine takes FASTA file formats as input, so all alignments files in the database were first converted to this format using the `seqret` program from the EMBOSS software package. Although we will work with truncated alignments, special care was taken to run DynaMine on the full sequences rather than the truncated ones. Indeed, protein flexibility is of course context-dependent (i.e. it does not depend on the local residue only, but also on the surrounding ones), so residues at the beginning and the end of the sequences should be accounted for when inferring the flexibility, even if terminal extensions will be discarded before aligning. An example of inaccuracies arising when running DynaMine on truncated sequences is portrayed on figure 2.2.4. Therefore DynaMine values were computed for the full sequences, then truncated, keeping only the core sequences.

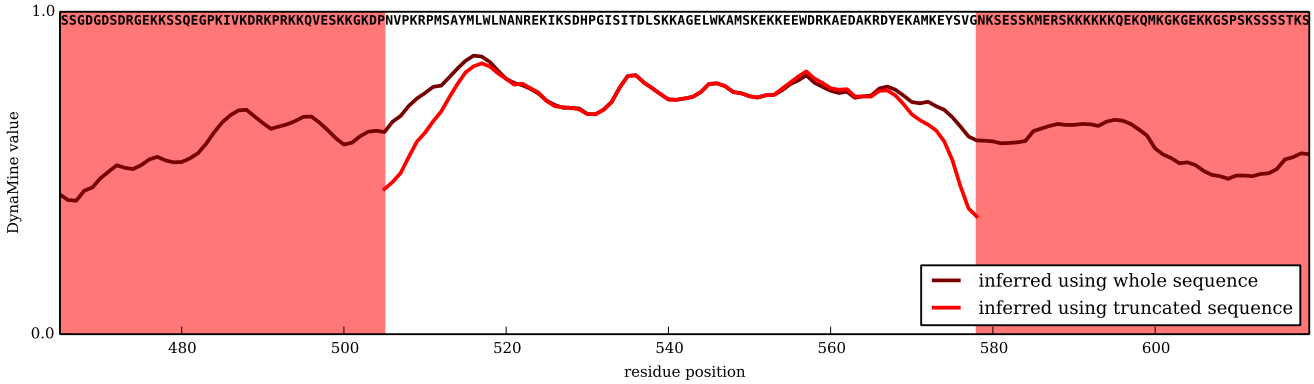

Figure 2.2.4: *DynaMine values computed using a whole sequence and computed using the corresponding truncated sequence. The correct way of using DynaMine is to run it on a whole sequence rather than on a subsequence.*

A DynaMine value of 1 means complete order (stable conformation), while a value of 0 means fully random bond vector movement (highly dynamic). However, since the values come from a linear prediction, DynaMine sometimes 'over-predicts' and outputs values outside of these bounds. We addressed this problem by simply capping all values to the $[0, 1]$ range. Furthermore, all values were binned into 50 equal-width bins, i.e. a DynaMine value of $x \in [0, 1]$ is replaced by the integer $\lfloor 50 \cdot x \rfloor$, allowing us to work with integer values (from now on these integer values will also be called 'Dynamine values').

At this point, for each multiple sequence alignment in BAliBASE we had one FASTA file (containing the actual alignment) and several text files (one for each sequence in the alignment) containing the (binned) DynaMine values. In order to minimize the time spent on writing text-handling computer code, the data on amino acid residues, DynaMine values, and inserted gaps were combined in one single text file per alignment, using a custom and easily-parsable file format.

In addition to aligning the benchmark data, we will also use it to infer the parameters used in the alignment algorithm. To avoid using the same data for both tasks, 50-50 cross-validation was used: we partitioned the alignments into a training set and a test set, the first one to be used for inferring parameters (such as dynBLOSUM matrices), and the second one providing the data to be aligned. Since alignments contained in a given BAliBASE dataset all share the same characteristics, how we partition the data is irrelevant. Therefore we chose to simply gather even-numbered alignments into one set and odd-numbered alignments into another set.

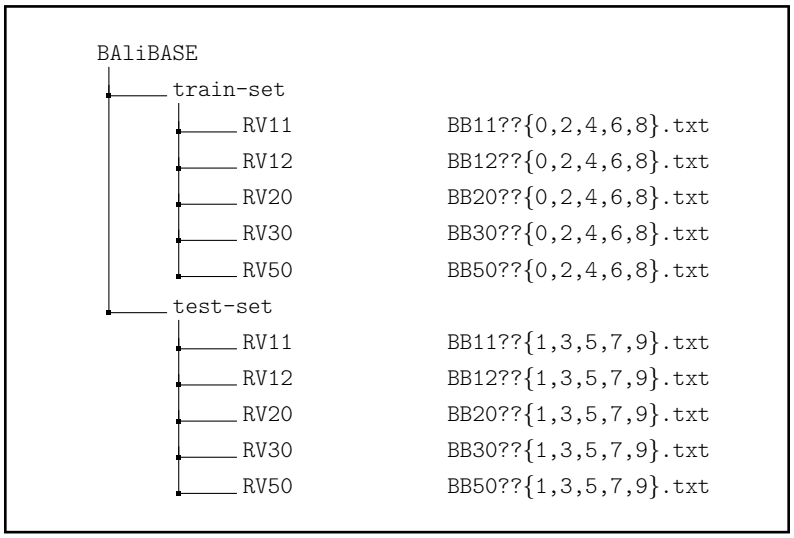

Figure 2.2.5: *Directory structure of the benchmark database used in this thesis. Next to each dataset folder is the UNIX pattern-matching string corresponding to its alignments.*

The final structure of our resulting benchmark database is summarized on figure 2.2.5. To give an idea of the computational challenge involved in aligning sequences in the test set, we counted that it contains a total of 79 947 pairs of sequences, each of which requires quadratic time (in sequence size) to be aligned. This will be time-consuming, even on a powerful computer.

2.3 Statistics about the predicted data

Before starting any experiment with our data, we thought it would be interesting to give some general statistics on the distribution of DynaMine values in the BAliBASE databases. Those are depicted on the three figures in this section. Figure 2.3.1 shows that most DynaMine values are around 0.7; in fact there was no value under 0.4 in our data. Figure 2.3.2 clearly shows that values decreased near the ends of a sequence; which justify our decision of truncating sequences in the previous section. Finally, figure 2.3.3 depicts the median, interquartiles, minimum and maximum values for each amino acid; but in our opinion they are mostly all the same.

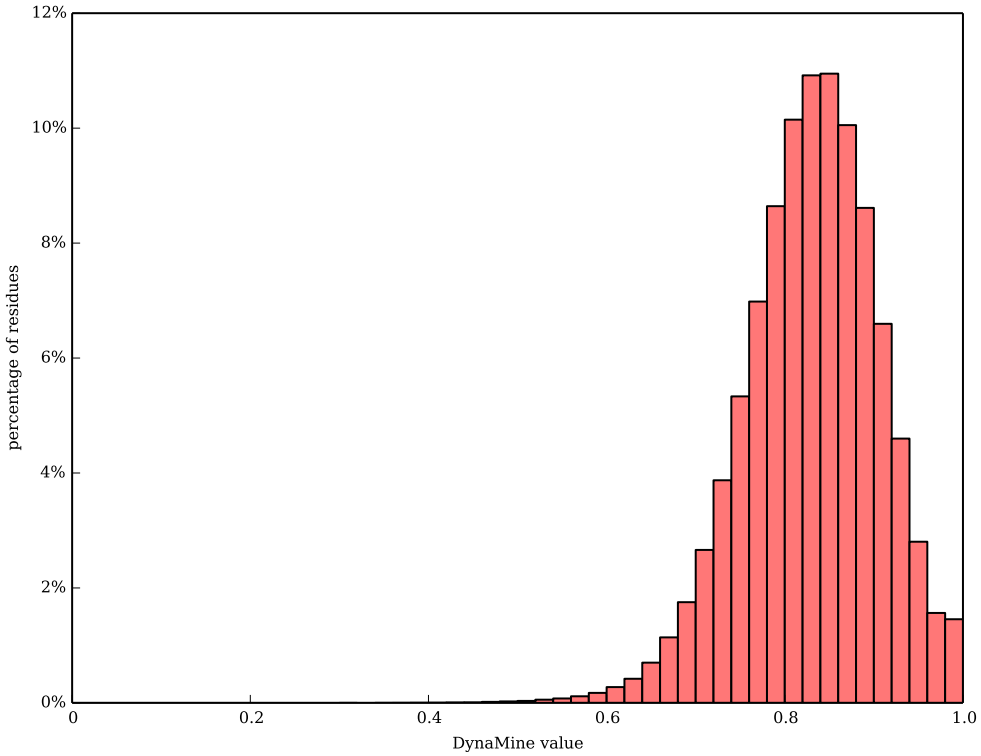

Figure 2.3.1: *DynaMine values distribution.*

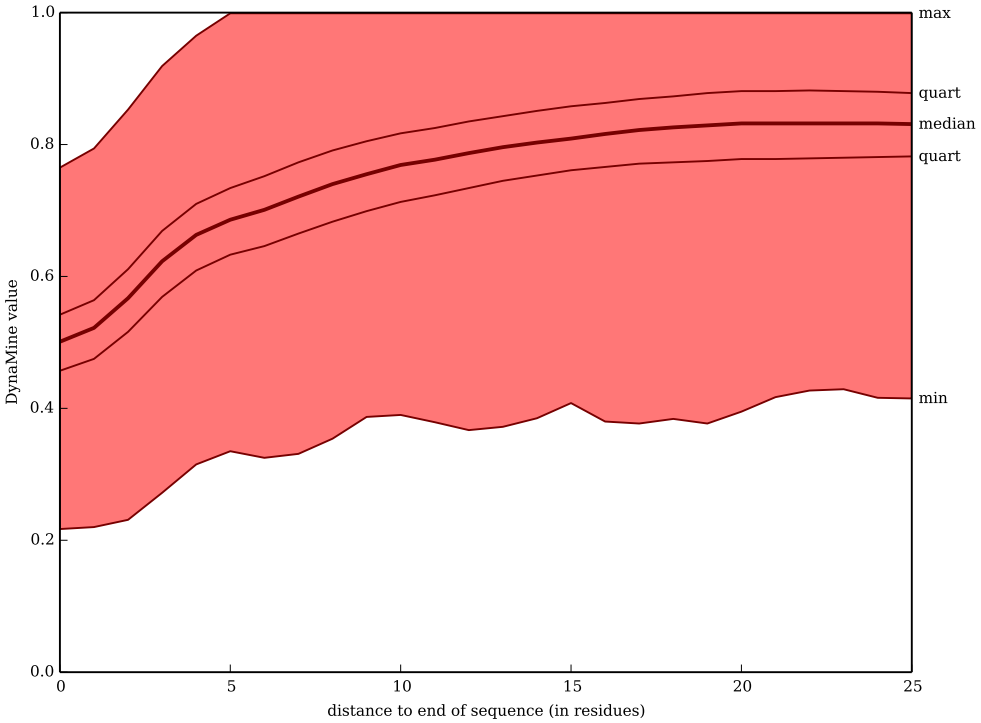

Figure 2.3.2: *DynaMine values near end gaps.*

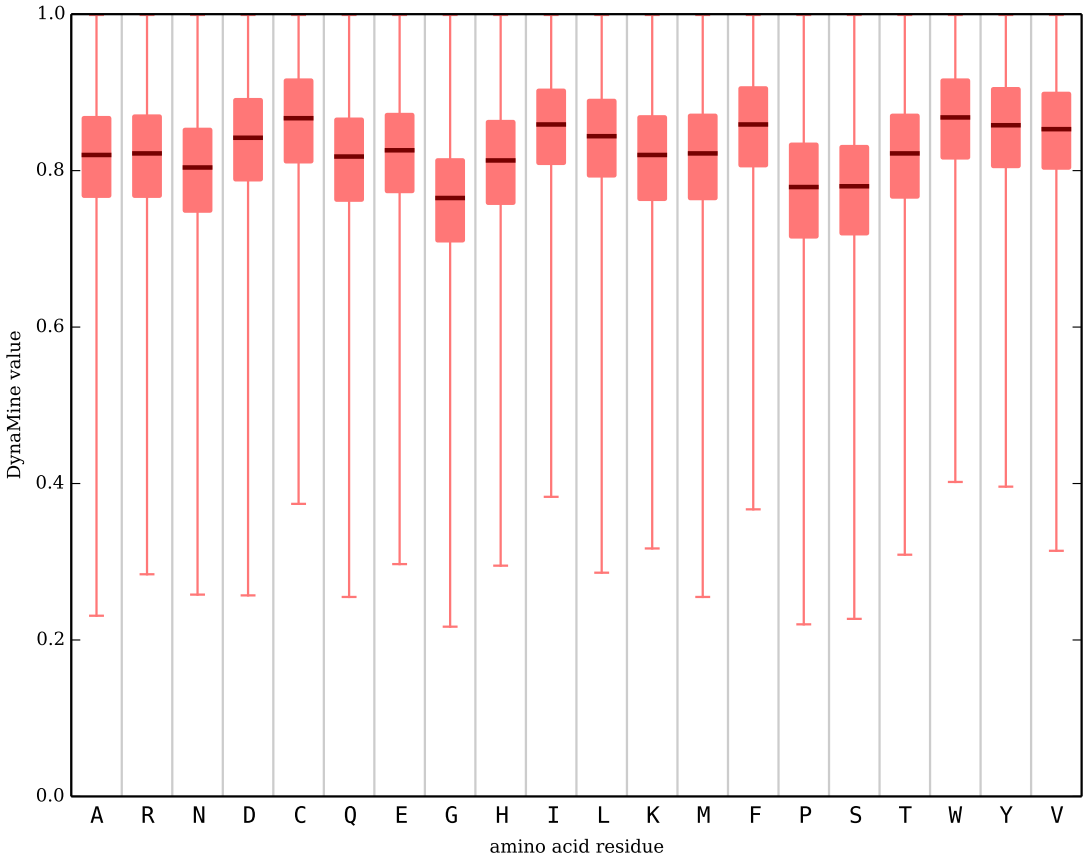

Figure 2.3.3: *DynaMine values for each amino acid residues.*

Chapter 3

Improving Needleman-Wunsch

3.1 Inferring the BLOSUM matrices

The algorithm for generating BLOSUM matrices, which we described in section 1.4, makes no assumption on the nature of the 'residues' in a sequence alignment. Therefore we may use the algorithm for inferring a 50×50 dynBLOSUM matrix rather than a 20×20 seqBLOSUM matrix. This is what will be done in this section; however, as a safeguard against implementation mistakes, we will first generate seqBLOSUM62 matrices and compare them with the standard BLOSUM62 matrices available in most bioinformatics software packages. Once assured that our computer code produces correct matrices, we will adapt it to work with DynaMine values.

Recreating seqBLOSUM62

As a first step we wrote a Python script which computes a seqBLOSUM matrix from a set of multiple sequence alignments, using the algorithm of section 1.4. Note that the algorithm needs gap-free alignments of same-length sequences. Therefore the script extract gap-free blocks from the alignments, by discarding columns containing a gap (see figure 3.1.1), and use the resulting set of blocks as its input.

```
NLFVALYDFV ASGDNT LSITKGEKLRVLLGY N--------------H NYGEWCEAAQTK N---------- GQGYWVPFSNYITPVN
YQYRALYDYK KEREED IDLHLGDILTVKNKG SLVALGFSDGQEARPE EQIGWLNAGYNE TTG-------- ERGNDFPFGTYVEYIG
NFRVYYRDSR ------ DPVWKGPAKLLLWKG ---------------- EQGAVVIAQDNS ---------- DIKFVVPFRRKAKIIR
FKVQAQHDYT ATDTDE LQLKAGDVVLVLIPF QN----------PEEQ DYEGWLMAGVKE SDWNQHKELEK CRGEVFPFENFTERVQ
EIAQVTSAYV ASGSEQ LSLAPGQLILIKLKK N---------------- TYSGWWQAGELQ ARGKK-----R QKGNWFPFASHVKLLG
```

Figure 3.1.1: *This multiple sequence alignment contains 4 gap-free blocks.*

Besides a set of benchmark alignments, our script also requires two parameters: the clustering threshold T and the scaling constant λ. We fixed $T = 62$ and for λ we used the same scaling constant that was originally used for creating the standard BLOSUM62

matrix used by common software such as EMBOSS or BLAST; its value can be found in (for example) the substitution matrix files available in the EMBOSS source code. Inferring a seqBLOSUM62 matrix from the BAliBASE database may of course produce a matrix very different than the one of EMBOSS, which was inferred from another benchmark database called BLOCKS and specifically created for the purpose of investigating amino acids substitution [HGPH00, HHP99]. Therefore the BLOCKS database was obtained from its server at `http://blocks.fhcrc.org/` and its alignments converted to a format readable by our script (BLOCKS comes in the form of a large single text file). The seqBLOSUM62 matrices inferred from BLOCKS and BAliBASE are pictured and compared with the standard one in figures 3.1.2 and 3.1.3.

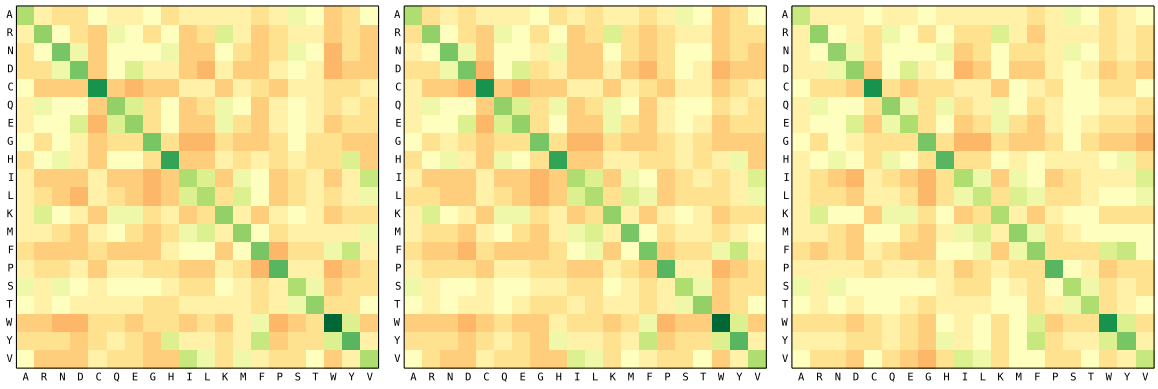

Figure 3.1.2: *Three seqBLOSUM62 matrices pictured as heat maps, from left to right: the standard one, the one generated from BLOCKS, and the one generated from BAliBASE. Scores range from* $+11$ *(dark green) to* -11 *(dark red, not present here).*

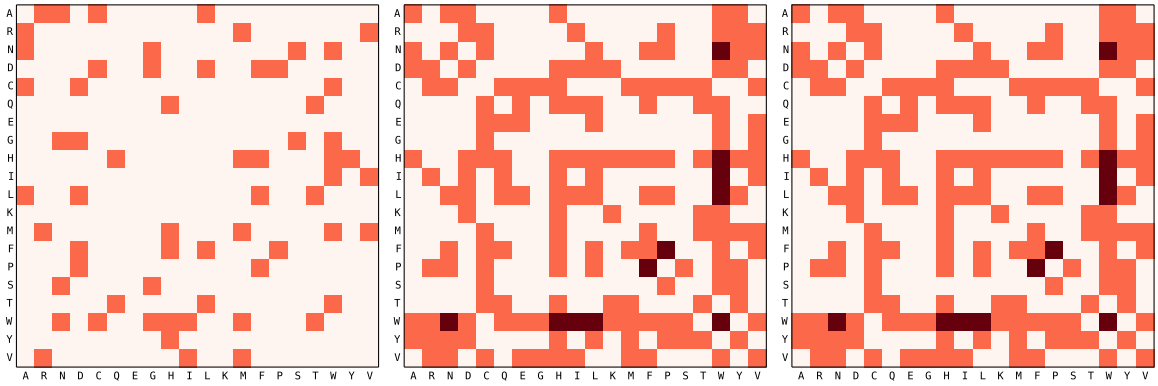

Figure 3.1.3: *Comparison of the seqBLOSUM62 matrices generated from BLOCKS (left) and BAliBASE (right) with the standard matrix used in most bioinformatics software. Score difference is* 0 *when white,* 1 *when red, and* 2 *when dark red.*

Counting the red cells of the first matrix on figure 3.1.3, we see that the seqBLOSUM62 matrix generated from the BLOCKS database differ from the standard seqBLOSUM62 matrix in 32 of its $1 + \cdots + 20 = 210$ substitution scores. This is consistent with the claim in [SJRS08] that the standard seqBLOSUM62 matrix is wrong in 15% of its

positions.

We thought it would be interesting to also compare our generated matrices with the 'correct' one (rather than the incorrect standard one). Our seqBLOSUM62 matrix generated from the BLOCKS database is again depicted on figure 3.1.4, together with the 'correct' matrix from [SJRS08] (which is available in the supplementary notes of the article). We see that in both matrices, differences with the standard BLOSUM62 matrix often occur at the same positions (but not always). There are still differences between our matrix and the 'correct' one: we counted that $12/210 = 5.7\%$ of the values do not match (but they never differ by more than 1). We think that these errors are due to different interpretations of the clustering method described in [HH92]; see section 1.4 for more information.

Whichever is the real correct BLOSUM62 matrix (if such a thing is even possible to define), we believe that our own version is likely close to it, and therefore the seqBLOSUM-generating program we implemented is probably mostly correct. Moreover, the [SJRS08] article claims that the errors in the standard BLOSUM62 matrix actually slightly improve alignment performance, therefore there is likely no reason to worry about minor miscalculations in our implementation.

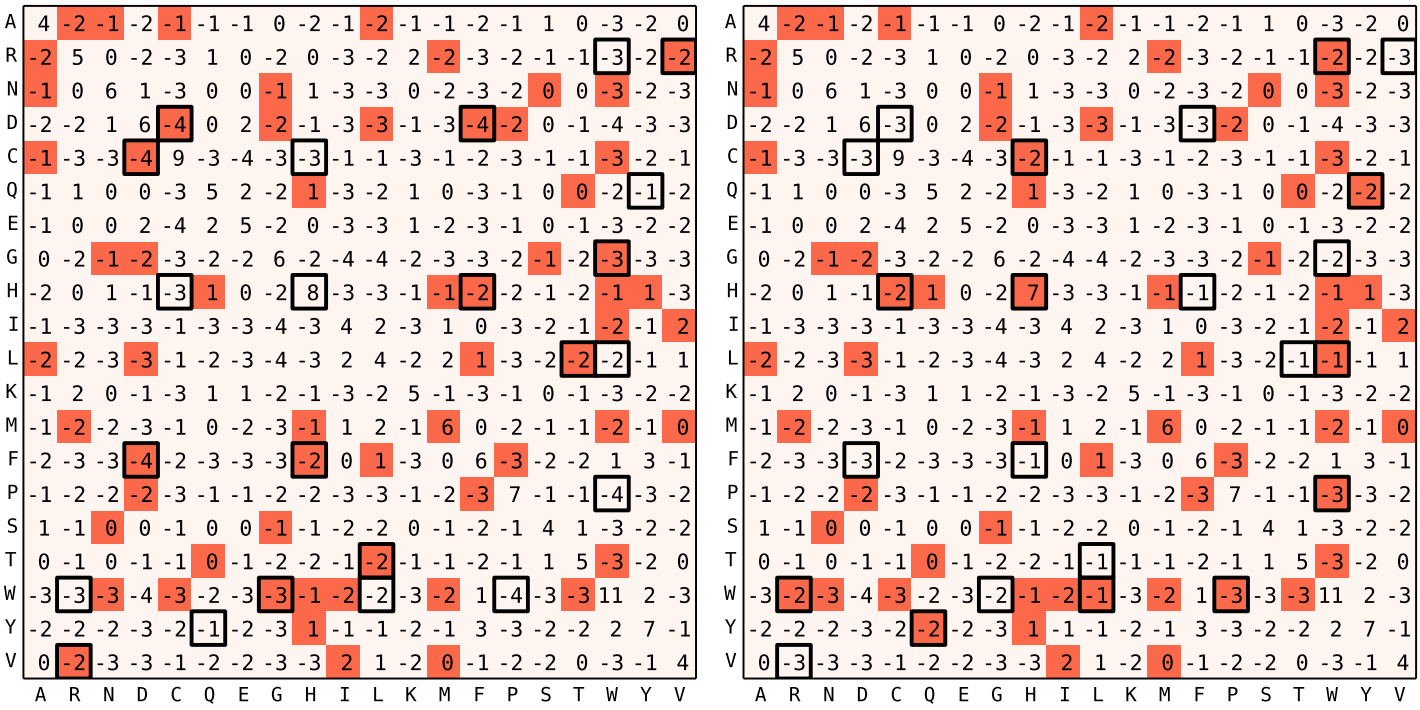

Figure 3.1.4: *Substitution scores in the seqBLOSUM62 matrix generated from BLOCKS (left), and in what is the 'correct' BLOSUM62 matrix according to [SJRS08] (right). Differences with the standard BLOSUM62 matrix are again depicted using red cells. Boxed cells have different values in the left and right matrices.*

Adapting for dynBLOSUM

Our Python script was then adapted to produce dynBLOSUM matrices. The algorithm stays the same, except that we now count pairs of DynaMine values rather than pairs of amino acid residues. However, caution is required, as it is possible that some pairs of values never appear in the data. Recall the definition of the scoring matrix:

$$s_{ij} := \frac{1}{\lambda} \log \left(\frac{q_{ij}}{p_i p_j} \right)$$

When there is no value i in the data, we have $p_i = 0$ and a division by zero. We take care of this situation by setting $s_{ij} = 0$ when $p_i = 0$ or $p_j = 0$. This makes sense: setting a zero score for DynaMine values never encountered simply means that we are neutral to any substitution involving such values.

Another problematic situation is when there are some values i and j in the data, but no pairs (i, j), so that we have $q_{ij} = 0$, $p_i p_j \neq 0$, and hence $s_{ij} = -\infty$. In this case, we set s_{ij} to be the minimum value (other than $-\infty$) in the substitution matrix.

Suppose we want to generate a dynBLOSUM62 matrix. So we must have $T = 62$, but how to choose the scaling constant? The scaling constant only purpose is to have 'nice' scores that can be rounded to integer values; it can be changed at will as long as we scale the gap penalties accordingly. However, our main experiment in this thesis requires the averaging of a seqBLOSUM with its corresponding dynBLOSUM. Having one matrix with large scores and the other one with small scores is therefore undesirable.

Our approach will be to choose the scaling constant λ of a dynBLOSUM so that the matrix has the same *expected* substitution score as the corresponding standard seqBLOSUM matrix. Since $p_i p_j$ is the probability of having a (i, j) pair at a random position in a random pairwise alignment, the expected substitution score is $\mu = \sum_i \sum_j p_i p_j s_{ij}$, and the desired λ can be computed from μ (which is -0.5209 for BLOSUM62, for example).

We can now use our script to infer dynBLOSUM matrices from any BAliBASE dataset and for any clustering threshold. See figure 3.1.5 for an example.

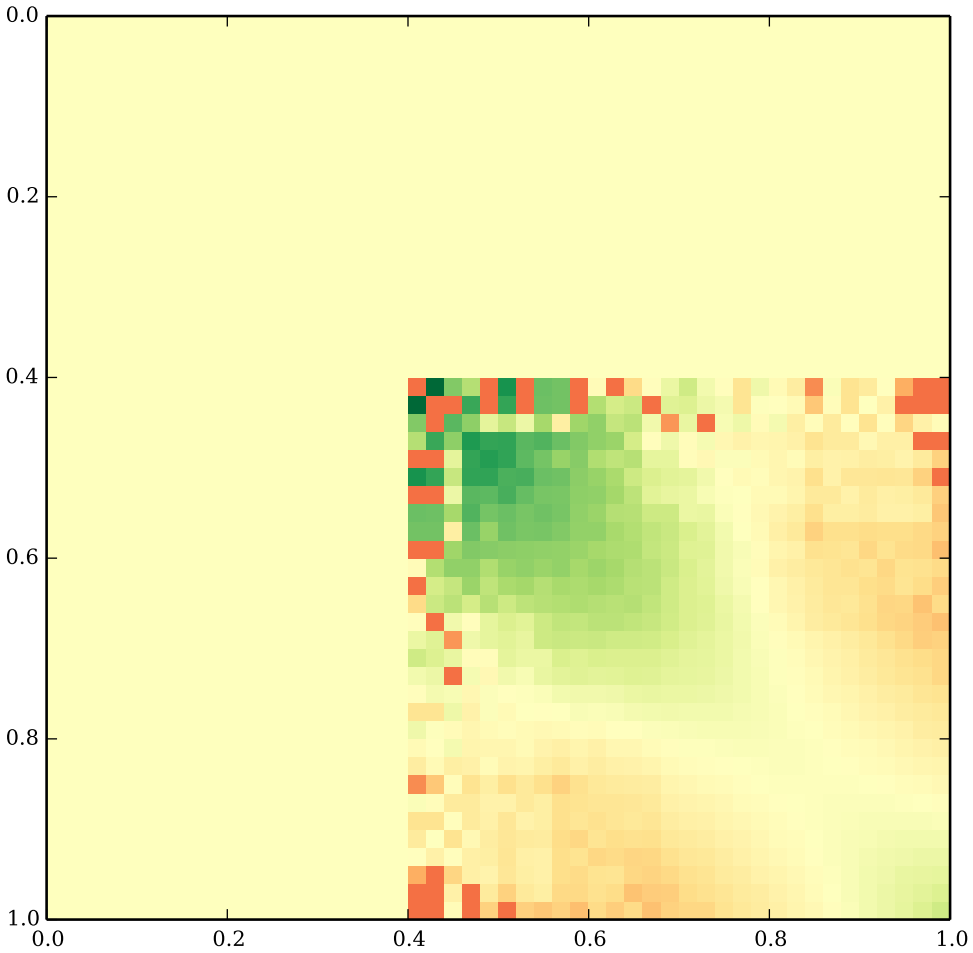

Figure 3.1.5: *A dynBLOSUM62 matrix generated from the RV30 dataset. There was no value under* 0.4 *in this dataset, hence the large 'empty' area.*

Remark. In this thesis we use the BAliBASE database as a benchmark but also as a train set for inferring parameters, such as BLOSUM matrices. Since the seqBLOSUM matrices were originally inferred from the BLOCKS database, it would be natural to do the same for dynBLOSUM matrices, i.e. we could run the DynaMine predictor on every sequence in the BLOCKS database, and then use the resulting alignments of DynaMine values to infer a dynBLOSUM matrix. However, most blocks in the BLOCKS database are quite short (often under 20 residues), consisting of highly conserved domains rather than complete sequences. Therefore running DynaMine on these short sequences will give untrustworthy values (see figure 2.2.4 in the preceding chapter).

3.2 Creating and scoring alignments

A small C library implementing the Needleman-Wunsch-Gotoh algorithm of section 1.3 was created for the thesis; information about where to obtain it is available in the

Appendix. We compared alignments produced with this library with the ones produced by the `needle` program from the EMBOSS software package, and got the exact same results, so we believe the implementation to be correct.

Once alignments are computed they still need to be compared to their benchmark counterparts. For this purpose we will use the sum-of-pairs score already described in section 1.4, for scoring individual pairwise alignments but also multiple alignments or whole datasets.

Aligners

All aligners used in our experiments were written using the aforementioned library. Most of them are small command-line applications, usually taking the following positional arguments:

- one seqBLOSUM matrix file (in NCBI/EMBOSS format)
- one dynBLOSUM matrix file (50 lines of 50 space-separated integer values)
- one or more numerical parameters, depending on what we want to investigate
- one (benchmark) multiple sequence alignment (in some custom file format)

For each pairs of sequences in the input alignment file, the program compute a pairwise alignment and print a line containing:

- the identifiers of the two sequences
- the number of correctly matched residues in the computed pairwise alignment
- the number of matched residues in the benchmark pairwise alignment

See figure 3.2.1 for an example of how it works. The program does not output the computed pairwise alignments, it just discards them instead. This is because at this point we do not really need to see the actual alignments: after all if it appears that some computed alignments have interesting sum-of-pairs scores, we can always recompute just those ones to examine them more closely. Moreover, this prevents us from having to store and organize almost 80 000 FASTA files for every tested combination of substitution matrices and numerical parameters.

In fact, with the BAliBASE database comes a small C program, called `baliscore`, which reads two alignments in the MSF file format and outputs its sum-of-pairs score. However using this ready-made program would require us to store every computed pairwise alignment, before converting them to MSF so that we can feed them to `baliscore`. We found that implementing the scoring function ourselves was simpler and quicker, so we did not use `baliscore`, although we compared its output with our program's output (scores were always identical).

```
$ ./aligner seqBLOSUM.txt dynBLOSUM.txt 50 BB11003.txt
1ad3_A          1uzb_A          337             413
1ad3_A          1eyy_A          209             397
1ad3_A          1o20_A          218             381
1uzb_A          1eyy_A          290             417
1uzb_A          1o20_A          221             337
1eyy_A          1o20_A          223             394
```

Figure 3.2.1: *Output of the program that will be used to compute and score pairwise alignments made by averaging two BLOSUM matrices (here with* $50\% - 50\%$ *weighting). First two columns are the protein identifiers, third column is the number of correct pairs in the computed pairwise alignment, and fourth column is the number of pairs in the benchmark pairwise alignment.*

Gap penalties. There are parameters other than substitution scores in a Needleman-Wunsch algorithm: the gap penalties. How to choose them is not clear [RP02, Mou08]. Here we chose to use penalties of 10 for opening gaps and 0.5 for extending gaps. Those are the default values of `needle`, the Needleman-Wunsch implementation from the EMBOSS software package.

On the other hand end gaps require special take. Since we work with truncated alignments devoid of end gaps, it makes no sense to let the aligners insert end gaps. So end gaps were forbidden, by setting very large gap opening penalties before the first residue and after the last residue.

Sum-of-pairs scores

Our aligners output scores for each pairwise alignment created. When faced with the problem of obtaining a total score for all pairwise alignments coming from one common multiple sequence alignment, we simply divided the sum of correctly aligned residues by the sum of benchmark aligned residues. So for example, in the case of figure 3.2.1, the total score is $\frac{337+209+218+290+221+223}{413+397+381+417+337+394} \approx 64\%$. We used the same method for obtaining a score for pairwise alignments coming from a whole BAliBASE dataset rather than a single multiple sequence alignment.

The rationale for this method is best described using graded school homeworks. Suppose you are in school, you have homeworks using a grading system akin to '12/20', but they are not all worth the same amount of points. So if you get a 10/20 for an important course and a 8/10 for a less important course, your average grade in percents is not $\frac{50\% + 80\%}{2} = 65\%$. Rather, it is $\frac{10 + 8}{20 + 10} = 60\%$. This is why we do not simply compute the total sum-of-pairs score as the average of the scores in percents: some pairwise alignments are longer than others, hence should make up for a larger part of the total sum-of-pairs score.

For further clarification, and as a future reference, our scoring method is more precisely summarized in figure 3.2.2.

Using the following notation:

- D for a benchmark dataset
- M for a benchmark multiple alignment (contained in some D)
- P for a benchmark pairwise alignment (taken from some M)
- $\hat{P}$ for the pairwise alignment computed from the two sequences in P

We define scores as follows:

$$\begin{aligned}
\text{pairs}(P) &:= \text{number of pairs in } P \text{ that are also in } \hat{P} \\
\text{total}(P) &:= \text{total number of pairs in } P \\
\text{score}(P) &:= \frac{\text{pairs}(P)}{\text{total}(P)} \\
\text{score}(M) &:= \frac{\sum_{P \in M} \text{pairs}(P)}{\sum_{P \in M} \text{total}(P)} \\
\text{score}(D) &:= \frac{\sum_{M \in D} \sum_{P \in M} \text{pairs}(P)}{\sum_{M \in D} \sum_{P \in M} \text{total}(P)}
\end{aligned}$$

Figure 3.2.2: *How sum-of-pairs scores are computed for a pairwise alignment, a multiple alignment, and a whole dataset. Recall that a multiple alignment of n sequences contains* $\frac{n(n-1)}{2}$ *pairwise alignments.*

Which seqBLOSUM is the best?

To ensure we had an efficient workflow in place before moving to more serious things, we aligned all the data in our test set, using every seqBLOSUM matrix available in the EMBOSS software package. Results are pictured on figures 3.2.3 and 3.2.4. Best results were attained with BLOSUM60 and worst with BLOSUM40, however the difference is thin. For the remainder of the thesis, we mostly use BLOSUM62 (the most commonly used substitution matrix in bioinformatics). Remark that results for the RV11 dataset are much worse than for the other ones. This is because RV11 is made up of highly dissimilar sequences (<20% residue identity), therefore using amino acid residues information alone is more susceptible of producing residue mismatches.

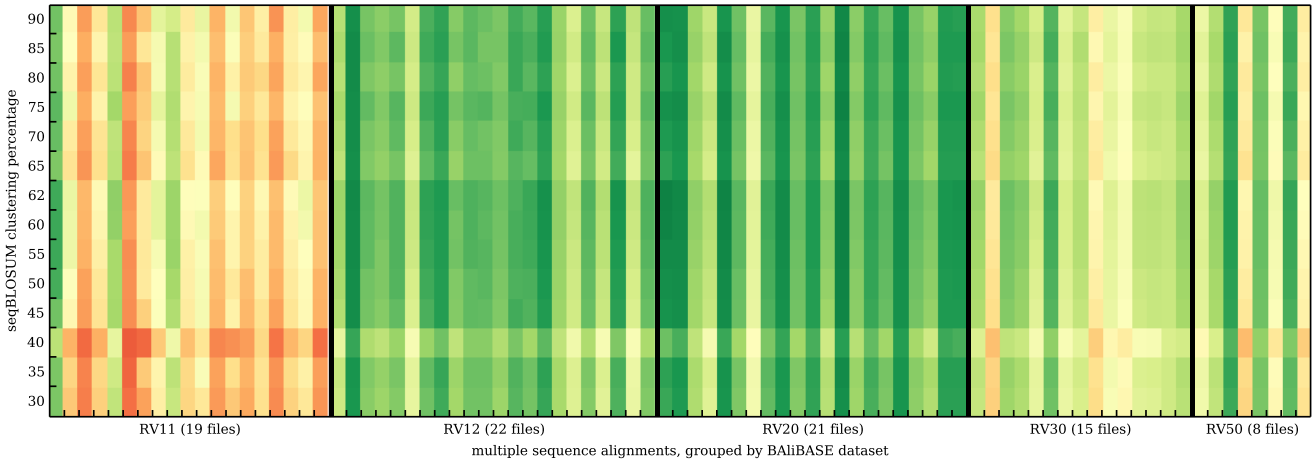

Figure 3.2.3: *Heat map of the (total) sum-of-pairs scores (red is 0%, green is 100%) for each BAliBASE alignment and each seqBLOSUM matrix in the EMBOSS software package.*

	30	**35**	**40**	**45**	**50**	**55**	**60**	**62**	**65**	**70**	**75**	**80**	**85**	**90**	
min	24	27	23	28	24	22	27	28	25	26	25	18	19	21	**RV11**
max	78	77	78	81	80	76	85	85	85	84	78	66	77	77	
mean	45	46	44	47	44	42	49	49	47	46	44	33	41	39	
total	35	38	36	39	35	34	41	40	39	37	35	26	32	31	
min	59	62	59	62	59	57	63	63	62	60	59	52	55	55	**RV12**
max	92	92	93	92	92	92	93	93	92	92	92	86	91	90	
mean	77	78	76	78	77	75	80	79	79	78	77	68	74	73	
total	75	77	75	77	75	74	79	78	77	76	75	65	72	71	
min	55	56	55	57	55	54	58	57	57	56	56	49	53	53	**RV20**
max	93	94	93	94	93	93	95	95	94	94	94	89	92	92	
mean	81	82	81	82	81	80	84	83	83	82	81	74	79	79	
total	85	86	85	86	85	84	87	87	87	86	85	79	84	83	
min	40	42	40	42	40	39	43	43	42	41	40	34	38	37	**RV30**
max	82	84	82	84	82	81	85	85	84	83	82	73	80	78	
mean	62	64	62	63	62	61	65	64	64	62	62	54	59	58	
total	59	60	59	61	59	58	62	62	61	60	59	51	57	56	
min	43	46	43	45	44	41	48	46	46	44	43	33	39	38	**RV50**
max	87	88	87	88	87	86	89	89	88	88	87	82	86	85	
mean	63	65	64	66	64	63	67	67	66	65	64	56	62	61	
total	59	61	59	61	60	58	63	62	62	61	60	52	58	57	

Figure 3.2.4: *Minimum, maximum, average, and total sum-of-pairs scores (in percentages) for each dataset and each seqBLOSUM matrix. By total score, we mean the one obtained by counting correct pairs across all multiple alignments in the dataset (see figure 3.2.2).*

An interesting observation is that the total dataset score is always worst than the average of the multiple alignments scores, except in the case of RV20 where it is the opposite and total score is always better than average score. This is probably because every RV20 alignment is made of sequences from a same protein family but aligned with one highly divergent sequence (see BAliBASE dataset descriptions on figure 2.2.1). The inclusion of

one very different sequence in each alignment decreases the score for this alignment (and thus the average multiple alignment score), but if we choose to look at all residue pairs across all alignments, then the few divergent sequences become less relevant and the total dataset score is better. This example shows us why it is important to investigate sum-of-pairs scores at all three levels (i.e. pairwise, alignment, and dataset scores; see figure 3.2.2).

3.3 Averaging seqBLOSUM and dynBLOSUM

We now have BLOSUM-type matrices for pairs of residues (seqBLOSUM) and for pairs of DynaMine values (dynBLOSUM). What remains is finding out a way to combine both kinds of matrices into one single substitution score function, so that the Needleman-Wunsch algorithm can use it to align sequences using information from both the residue and the DynaMine value sequence. The main combination method used in this thesis is averaging.

Recall (see section 1.3) that the substitution score for aligning position i (in the first sequence) with position j (in the second sequence) was noted $\text{sub}(i, j)$. We will set it to the following weighted average of seqBLOSUM and dynBLOSUM:

$$\text{sub}(i,j) \;:=\; \alpha \cdot \text{seqBLOSUM}(x_i, y_j) \;+\; (1-\alpha) \cdot \text{dynBLOSUM}(u_i, v_j)$$

Where x_i, y_j and u_i, v_j are respectively the amino acid residues and DynaMine values at position i and j in each sequence. The number α is the proportion of seqBLOSUM: it ranges from 0.0 (pure dynBLOSUM alignment, amino acid residues are ignored) to 1.0 (pure seqBLOSUM alignment, so just like in common implementation of Needleman-Wunsch). Also recall that in section 3.1 we generated the dynBLOSUM matrices so that they have the same expected substitution score than their seqBLOSUM counterparts. Therefore, in theory, gap penalties can stay the same when varying the value of α. Alignments parameters used for the experiments in this section are summarized on figure 3.3.1.

seqBLOSUM	standard BLOSUM62 matrix
dynBLOSUM	same clustering and expected score as the seqBLOSUM
gap penalties	10 (opening) and 0.5 (extending)
end gaps	disallowed

Figure 3.3.1: *Alignments parameters used in the experiments.*

Using the aligner program described in the previous section, we computed sum-of-pairs scores for all pairs of sequences in our BAliBASE test set. If it is not explicitly stated otherwise, we are always using a 62% clustering percentage (see discussion surrounding figure 3.2.3. The α parameter was varied in 0.01 increments, which required several

millions of pairwise alignments to be computed, each of whose costs quadratic time in sequence time, so the experiment took a quite long computation time, especially if we factor in all the inevitable first attempts which failed because of errors in data preprocessing, algorithm implementation, parameter choices, and so on.

Results for our BAliBASE test datasets

The relationship between the α parameter and the total sum-of-pairs score for each BAliBASE dataset is plotted on figure 3.3.2. Of course, we generated (from the training sets) a different dynBLOSUM matrix for each of the test datasets. Scores can be computed for pairwise sequence alignments (PSA), for multiple sequence alignments (MSA) and for the whole dataset (see scoring methods summary on 3.2.2). For each dataset, we plotted the median PSA score, the average (arithmetic mean) MSA score, and the total dataset score. We chose to use the median for PSA scores because there are usually many PSAs in a given dataset, many of which are extreme cases. Since there are only a few MSAs in each dataset, we rather used the mean for these.

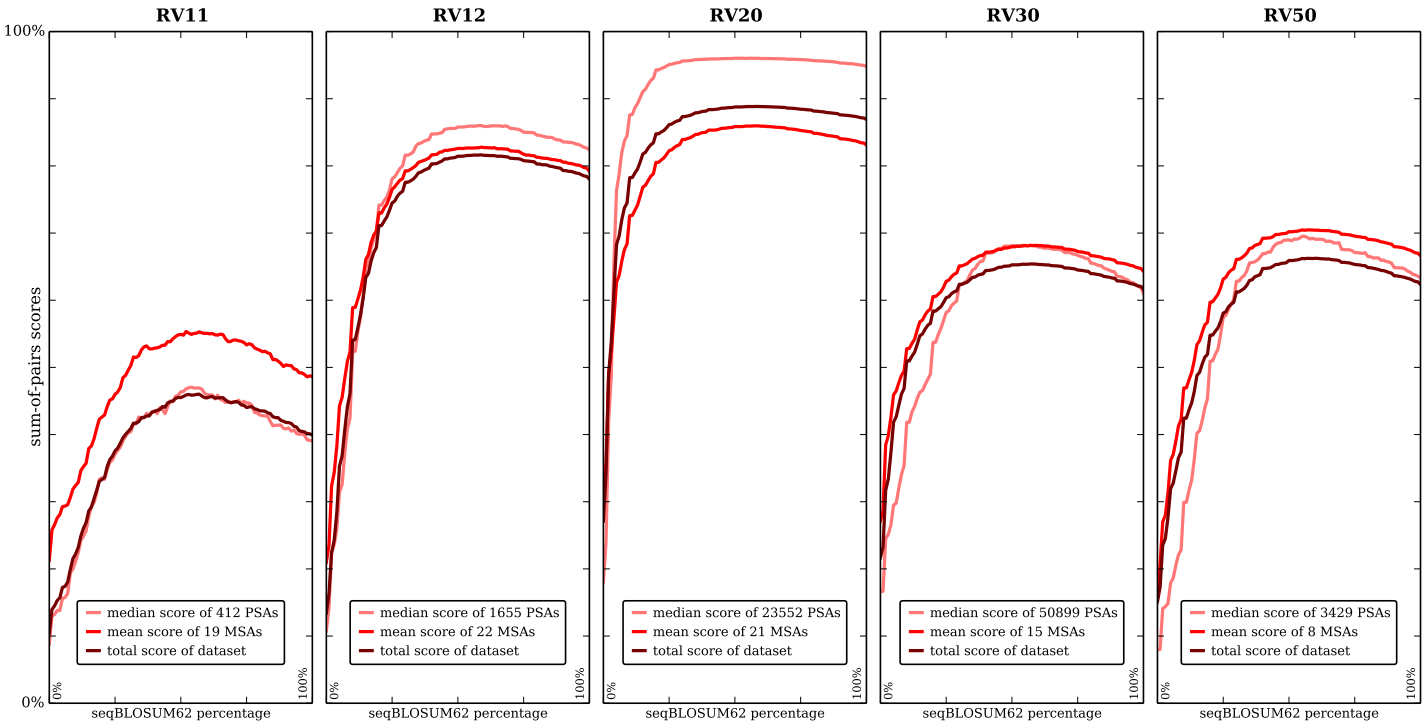

Figure 3.3.2: *Median PSA score, average MSA score, and total dataset score for α going from* 0 *(pure dynBLOSUM62) to* 1 *(pure seqBLOSUM62), shown for each BAliBASE test set.*

Some observations can already be made: for example, best improvements occur in the RV11 and RV12 datasets, which contain alignments with low similarity; we will talk about that in details in a later subsection. Also, the total dataset score is always lower than the mean MSA score, except in the case of RV20: this was already explained in the discussion following figure 3.2.4, this is probably because all alignments in RV20

have many similar sequences plus one very different 'orphan' sequence. These many similar sequences in each RV20 alignment also explains why the improvement is low in this dataset: another link between similarity and possible improvements.

Results of particular interest are summarized in figure 3.3.3 (using the total dataset score only). Since alignments were also computed for BLOSUM30, their scores are also depicted on the table.

BAliBASE dataset	**RV11**		**RV12**		**RV20**		**RV30**		**RV50**	
clustering percentage	**30**	**62**	**30**	**62**	**30**	**62**	**30**	**62**	**30**	**62**
best α value	0.50	0.57	0.53	0.59	0.50	0.58	0.50	0.58	0.50	0.57
best score	43%	46%	81%	82%	88%	89%	64%	65%	64%	66%
pure seqBLOSUM score	35%	40%	75%	78%	85%	87%	59%	62%	59%	62%
pure dynBLOSUM score	11%	10%	14%	13%	27%	27%	22%	22%	15%	15%

Figure 3.3.3: *Best total dataset scores, with the corresponding α values, when averaging both seqBLOSUM and dynBLOSUM, compared to scores obtained when using only one of each BLOSUM matrix (i.e. when setting $\alpha = 1$ or $\alpha = 0$).*

Unsurprisingly, aligning sequences using DynaMine information alone gives bad results. When averaging however, it is usually possible to obtain a slight increase in the sum-of-pairs score (the best increase being 6% for the RV11 dataset; we will talk about it again later). What is particularly noticeable is that the best seqBLOSUM percentage (i.e. the best α value) seems to stay the same across all datasets: around 50% for BLOSUM30 and 60% for BLOSUM62.

Keep in mind that scores in the two previous figures are total sum-of-pairs scores for whole datasets, all having different sizes and shapes. Therefore the improvements readable from figure 3.3.3 are only rough estimates of what happens at the individual pairwise alignment level. We need to zoom in on this data.

In the remaining of the section we will measure improvements by using the difference between the score with the overall best α (for the given dataset, see previous table) and the score with a pure seqBLOSUM matrix ($\alpha = 1$). For example an improvement of 10% means that a sum-of-pairs score went from 20% to 30% (and not to 22%!). It also means that 10% of the pairs in the benchmark were obtained with the averaging method, but were not obtained with the pure seqBLOSUM method.

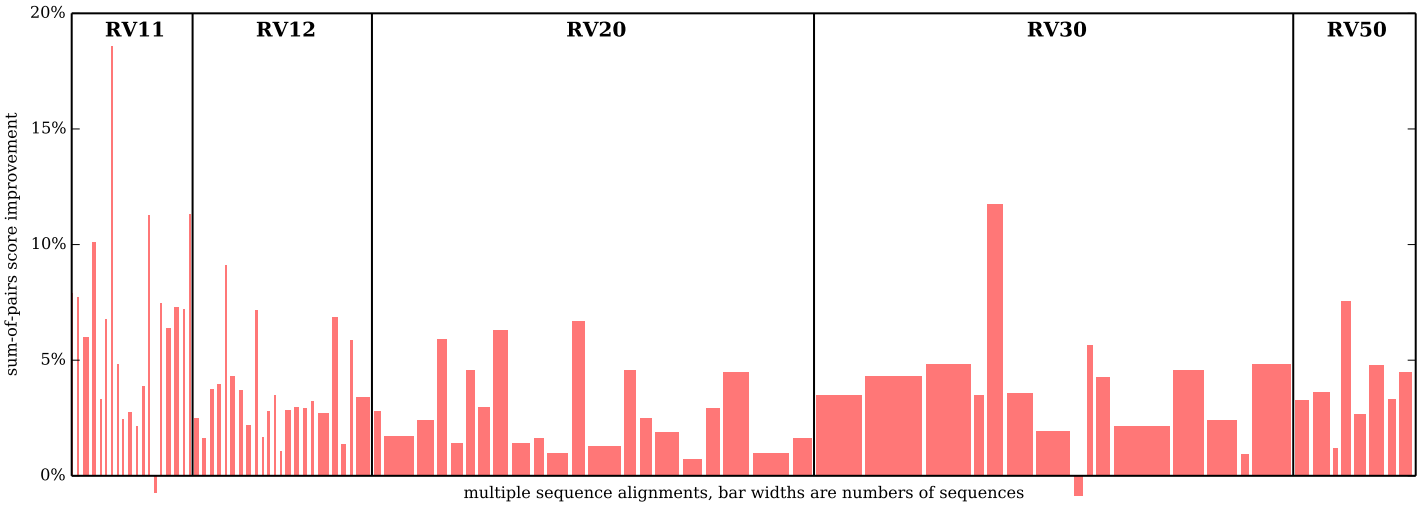

Figure 3.3.4: *Increase in the sum-of-pairs score of each multiple sequence alignment, grouped by dataset. Bar width indicate the number of sequences in an alignment.*

On figure 3.3.4, score improvements are showed on a multiple sequence alignment level. Since the number of sequences in a multiple alignment is relevant (an improvement is more worthy of attention if it occurs when considering many pairwise alignments), we use bar widths to indicate alignment sizes. A first encouraging observation is that scores are improved in nearly all multiple alignments, with only two exceptions in datasets RV11 and RV30. However, the largest improvements occur in the smallest alignments (see RV11 especially) and are therefore probably less significant.

Zooming in again on the data, we can look at improvements at the pairwise alignment level. However because of the large number of pairwise scores to consider (around 80 000), an accurate data visualization becomes difficult. Bar plots for each dataset were still attempted on figure 3.3.5, with one bar for each pairwise alignment (bar widths are now irrelevant), but in the larger datasets pairwise alignments are so numerous that we can no longer clearly distinguish the different bars. One observation we can still made from this plot is that the improvements pictured in figure 3.3.4 were in fact a bit too optimistic. This is particularly clear for the RV30 dataset, whose pairwise alignment score improvements seem to be often negative, although its multiple alignment score improvements were mostly positive.

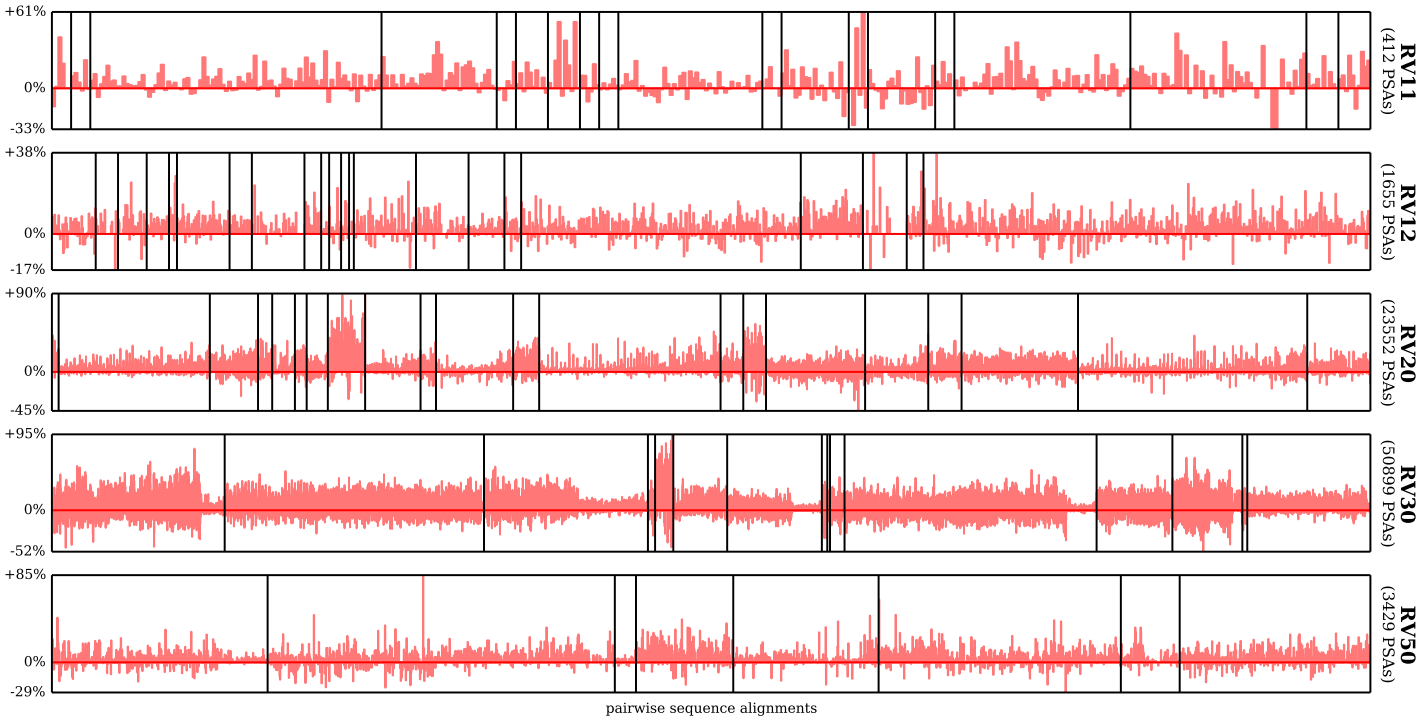

Figure 3.3.5: *Improvement of the sum-of-pairs score in each pairwise sequence alignment, grouped by the source multiple alignment.*

This observation brought us to create yet another plot, similar to figure 3.3.4, but this time showing the median pairwise score improvement for each multiple sequence alignment (figure 3.3.6). Many supposed improvements now disappear, for example in the RV20 dataset the median pairwise improvement is mostly zero. Still, for most pairs of sequences our averaging method is not worst than the pure seqBLOSUM method: the only two multiple alignments where median pairwise scores have decreased are in RV11.

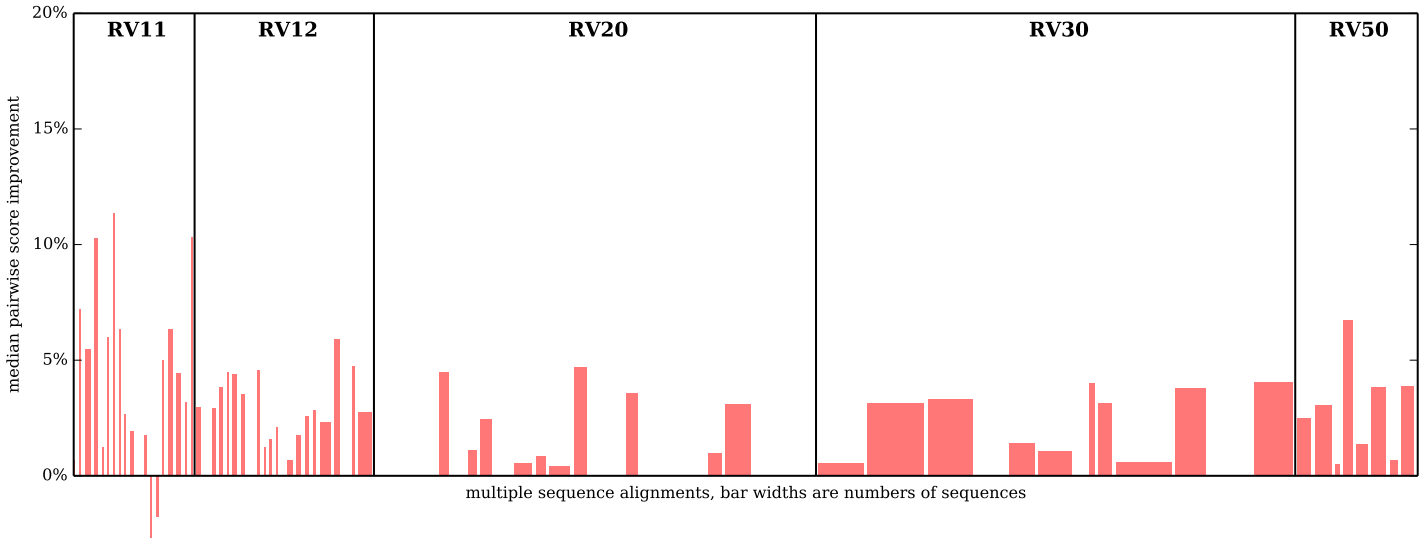

Figure 3.3.6: *Median increase in pairwise score for each multiple sequence alignment. Bar width indicate the number of sequences in an alignment.*

More details on improvements inside the RV11 dataset are available on figure 3.3.7. The two multiple alignments with decreased median pairwise scores are BB11025 and BB11027.

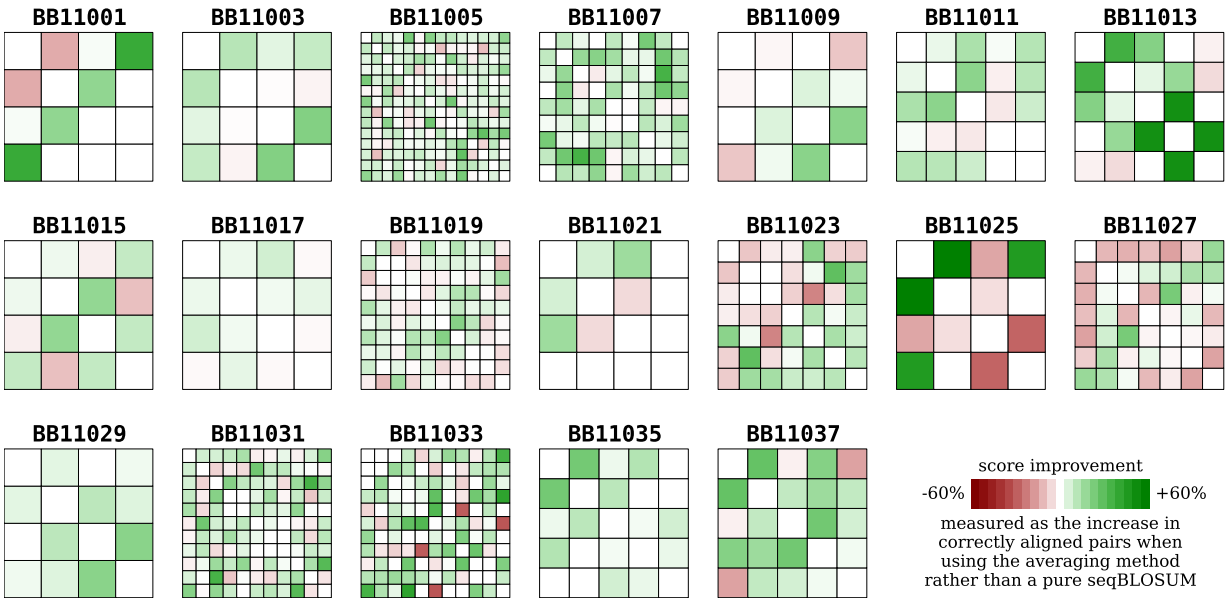

Figure 3.3.7: *Details of the sum-of-pairs score improvements for the RV11 dataset, when $\alpha = 0.57$ and with 62% clustering percentage. There is one matrix for every multiple sequence alignment in RV11, itself containing one cell for every pairs of sequences in the alignment. Hence matrices are symmetric and cells on the diagonals are always blank.*

Improvement and sequence similarity

Taking a look again at figure 3.3.6 we see that largest improvements reside in the RV11 and RV12 datasets. According to the BAliBASE documentation, these datasets contain sequences with low residue identity between themselves (see the description on figure 2.2.1). This is probably not a coincidence: if two sequences are highly dissimilar, it makes senses to use some information other than the amino acid residues (such as DynaMine values) when trying to align them. Motivated by this idea, we investigated the link between sequence similarity and score improvement when using a dynBLOSUM matrix.

We used two different measures of sequence similarity. The first one is the most obvious: the percentage of identical matched residues in the benchmark pairwise alignment. That is, we take a pairwise alignment from BAliBASE, count the number of pairs of identical amino acids, and divide it by the total number of pairs in the alignment. For the second similarity measure, we used the seqBLOSUM62 matrix in order to get a substitution score for each matched pair, summed all of these local scores, and then again we divided by the total number of pairs. We computed these two numbers for all 80 000 pairwise alignments in our BAliBASE test set, and compared them with the corresponding score improvements by using scatter plots on figure 3.3.8.

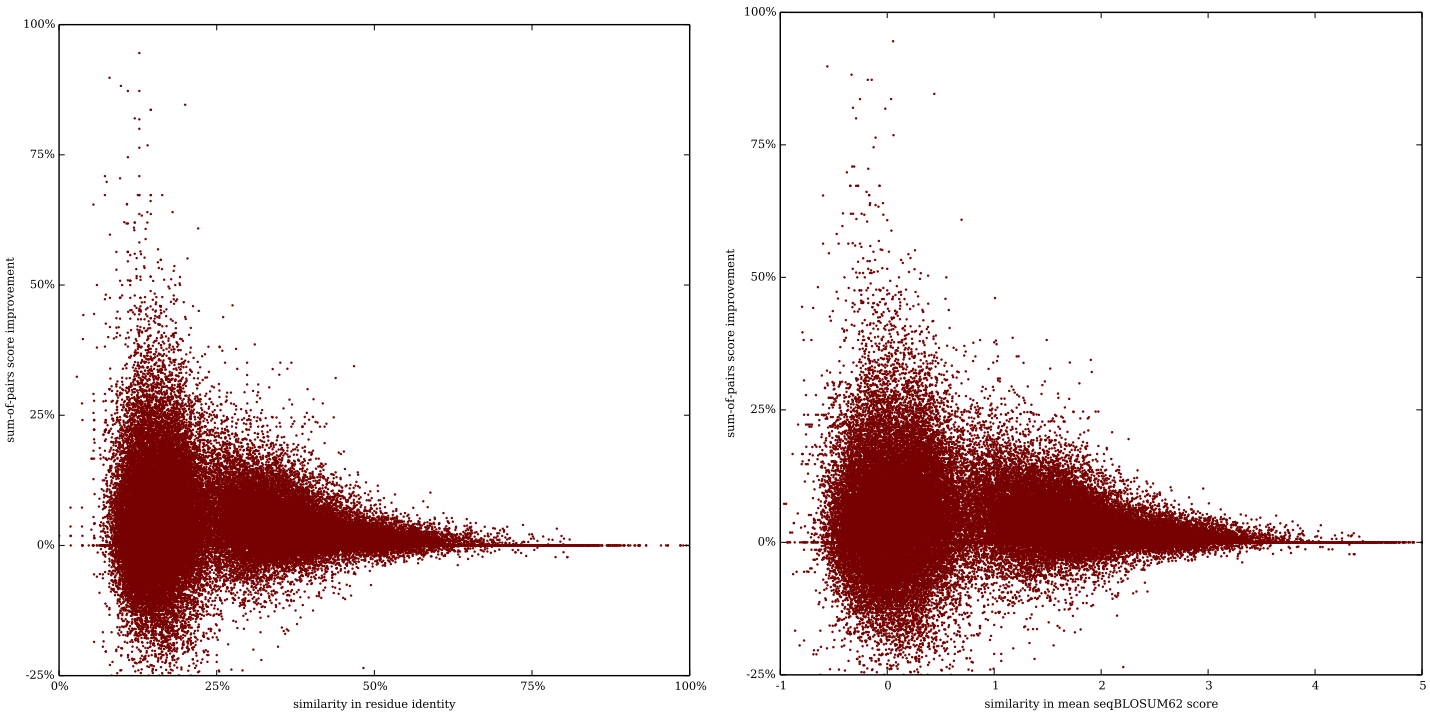

Figure 3.3.8: *Relation between pairwise score improvement and pairwise sequence similarity. There is one dot for every pairwise alignment in the BAliBASE test set.*

Both best and worst score improvements tend to occur with dissimilar pairwise alignments, however the very best ones mostly occur when similarity is lower than 25% (in

residue identity) or 1.0 (in mean seqBLOSUM62 scores). We should keep in mind that DynaMine uses the residue sequence to predict flexibility, therefore low similarity can also worsen DynaMine-based alignments.

Still, we wanted to further investigate possible improvements when using the averaging method on dissimilar sequences. Another experiment was set up: we took all pairwise sequence alignments in both RV11 and RV12 datasets (those contain equi-distant sequences with less than 40% identity), measured their similarity (in residue identity percentage), and randomly partitioned them into a training set and a test set. In order to avoid extreme cases arising from very short sequences, we also removed all pairwise alignments having less than 150 matched residues.

With the training set, we generated a dynBLOSUM matrix, but without using any clustering this time (after all, we already know that these sequences have low similarity); though we still normalized the matrix so that it has the same mean score as the seqBLOSUM62 matrix (see section 3.1). We then aligned every pair of sequences in the test set, for α going from 0 to 1, keeping track of the best α value and best corresponding score improvement. When several α values yielded the same improvement, we always picked the largest α, this way we avoid being overoptimistic about our averaging method.

Results were then ordered by sequence similarity, taking for each identity percentage the mean best *alpha*, the mean best improvement, and the total best improvement (the one obtained by summing all correct pairs, all benchmark pairs, and dividing the two numbers to get a percentage): see figure 3.3.9.

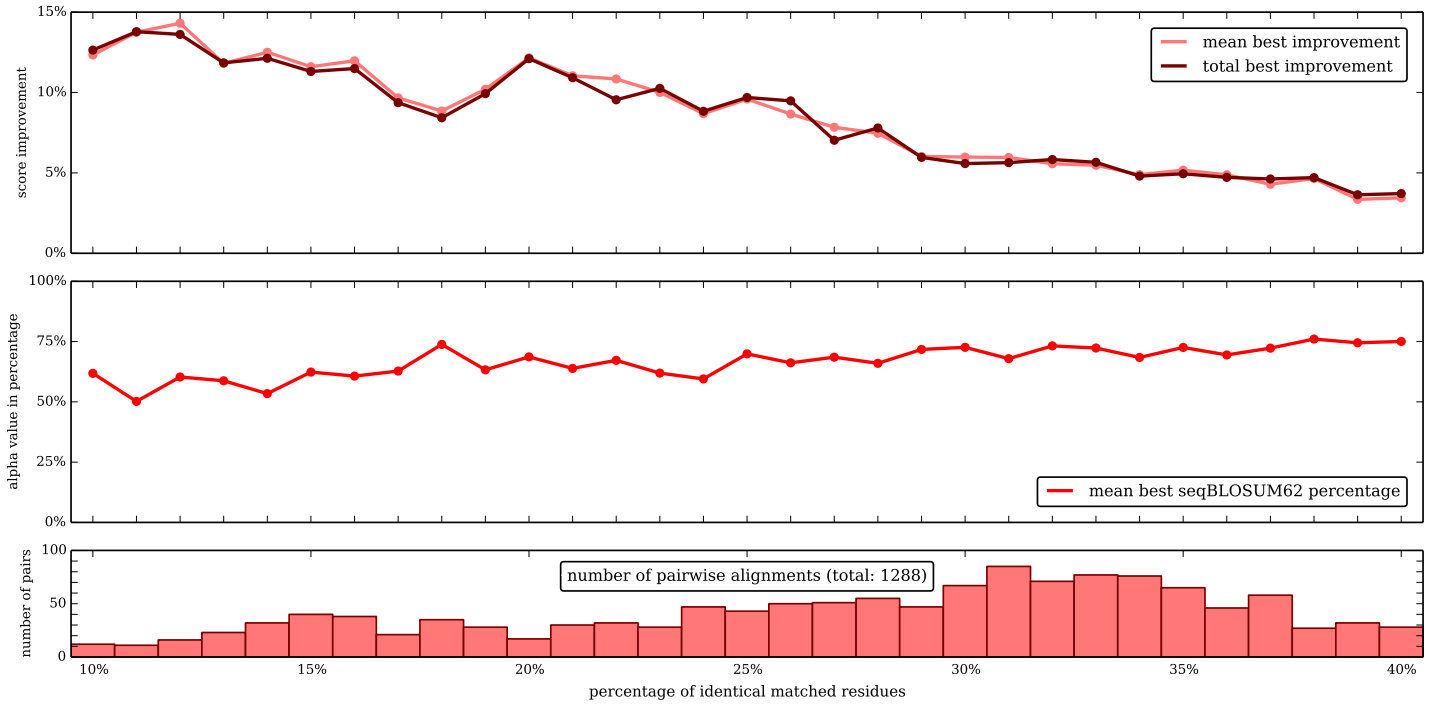

Figure 3.3.9: *Best possible improvements and corresponding best α values for each pairwise alignment similarity. Data generated using RV11 and RV12 datasets only.*

As we hoped, it seems that best possible improvements occur for more dissimilar pairwise alignments. The best α value (i.e. the percentage of seqBLOSUM used for aligning) also seems to slightly increases with similarity. However it should be noted that this experiment was conducted with a very small dataset: there are only 1288 pairwise alignments in our RV11+RV12 test set, and we further partitioned them by similarity. For this reason figure 3.3.9 also includes a bar plot showing how much pairwise alignments there are for each identity percentage (and we did not show the results for identity percentages concerning less than 10 pairwise alignments). It would be interesting to conduct the same experiment with a larger dataset of dissimilar pairwise alignments, maybe coming from a benchmark database other than BAliBASE, but unfortunately we had no time for such thing.

3.4 Other DynaMine-based scoring methods

Variations on averaging dynBLOSUM and seqBLOSUM matrices, or completely different ways of using DynaMine values in Needleman-Wunsch, were considered. We tried them, but not extensively, and because of computational time constraints, only with the (small) RV11 and RV12 datasets. We still think it is interesting to briefly list these other methods, as it is possible that some of them could give better improvements than those obtained in the preceding section.

Binning the DynaMine values differently.

In our averaging method, we first binned the DynaMine values into 50 equal-width bins. We tried using different numbers of bins, but this did not lead to significantly better score improvements. In the extreme case of 1000 bins (which means, no bins at all, since DynaMine values in $[0, 1]$ are only predicted up to three digits after the decimal point), we obtained very bad results. This is unsurprising since many DynaMine values never appear in our training data (hence having zero scores in the inferred dynBLOSUM matrix), although they could appear in the testing data. The other extreme of using very few bins (say, 10 bins or less) was of course also uninteresting. Another way of binning, which seemed a good idea at first, was to use equal-frequency bins, but unfortunately this did not yield better results either.

Scaling the dynBLOSUM matrix differently.

When generating a dynBLOSUM matrix, we scaled its values so that its expected score for random pairwise alignments was the same as the one of the corresponding seqBLOSUM matrix. We could have chosen another way of scaling the matrix, but then default gap penalties would probably have to be changed as well, and we did not know how to find the correct ones.

Two weighting parameters

Instead of using matrices weights summing to 1, we can also simply use two different unrelated weighting coefficients:

$$\text{sub}(i,j) \;:=\; \alpha \cdot \text{seqBLOSUM}(x_i, y_j) \;+\; \beta \cdot \text{dynBLOSUM}(u_i, v_j)$$

An advantage of this approach is that we no longer have to worry about the choice of the scaling constant for the dynBLOSUM matrix (this constant is built-in into the β coefficient). But since we then have to try (α, β) pairs of coefficients, the number of pairwise alignments to compute becomes quadratic rather than linear in the number of pairs of sequences. It is therefore much more time-consuming, which prevented us of investigating this approach more closely. But for the very few (α, β) pairs that we did try, we did not notice any significant improvement than when simply setting $\beta = 1 - \alpha$.

Using dynBLOSUM scores as scaling coefficients.

Rather than averaging seqBLOSUM and dynBLOSUM matrices, we can use the scores in the second matrix as proportions by which to increase (or decrease) the scores in the first matrix. More formally, the scoring function used is:

$$\text{sub}(i,j) \;:=\; \text{seqBLOSUM}(x_i, y_j) \;+\; \frac{\text{dynBLOSUM}(u_i, v_j)}{\alpha} \cdot |\text{seqBLOSUM}(x_i, y_j)|$$

The idea is easier to understand using an example. Suppose that $\alpha = 100$ and that for a given pair of positions (i, j), the corresponding dynBLOSUM score is -20. Then it means that the sub(i, j) score will be the corresponding amino acid substitution score from seqBLOSUM, but decreased by 20% of this value. Varying the α parameter then increases or decreases the effect of the dynBLOSUM matrix. However, when testing this approach, it seemed that best alignments were always for large values of α, in which case we just have a pure seqBLOSUM substitution matrix.

An unified mixBLOSUM matrix.

A more general way of using both amino acid residues information and DynaMine values is to infer a single large BLOSUM matrix with scores for every pair of (acid, value) pairs. This means that instead of having one 20×20 seqBLOSUM for the 20 amino acids and one 50×50 dynBLOSUM for the 50 DynaMine bins, we use a 1000×1000 'mixBLOSUM' for the $20 \times 50 = 1000$ (acid, value) possible pairs:

$$\text{sub}(i,j) \;:=\; \text{mixBLOSUM}\,((x_i, u_i), (y_j, v_j))$$

Unfortunately we are not sure that the BAliBASE database contains enough data to infer such a mixBLOSUM matrix, so this approach was not investigated.

Dynamic Time Warping.

Although nothing interesting arised from this approach, we believe it is worth mentioning, since it was one of the first experiment done for this thesis. DynaMine values can be seen as a continuous signal (when we do not use bins), so it makes sense to try a signal-matching algorithm such as Dynamic Time Warping. However, we could not get good alignments with DTW, we believe it is mainly because there is no gap opening penalty in DTW, so it tends to insert many small gaps everywhere, something which is biologically undesirable. Once we discovered that DTW was just a special case of Needleman-Wunsch-Gotoh (see section 1.3), we abandonned the idea of using it. Figure 3.4.1 is one example of DTW alignment.

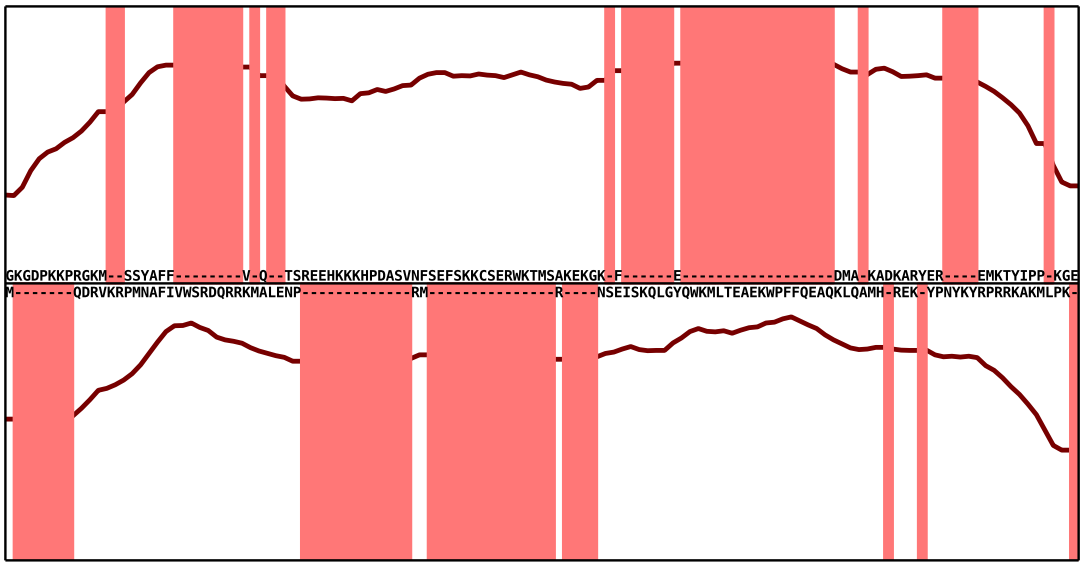

Figure 3.4.1: *Pairwise sequence alignment created by matching the DynaMine signals with DTW (using Euclidean distance).*

3.5 Conclusion

In conclusion, we list what we learned in this thesis, in chronological order.

Needleman-Wunsch-Gotoh.
The usual algorithm, as described in [DEKM98], can easily be generalized to allow for position-dependent substitution scores and gap penalties. Moreover, this generalization permits us to see Dynamic Time Warping as a special case of Needleman-Wunsch-Gotoh.

Clustering for BLOSUM.
The original BLOSUM article [HH92] is not very clear when describing the clustering method used. Our own interpretation was described in section 1.4 but maybe it is not the right one. This can results in different clusters, and therefore in slightly different BLOSUM matrices.

Recreating seqBLOSUM62.
The matrix we inferred from the BLOCKS database is slightly different than the classical seqBLOSUM62 matrix. The article [SJRS08] claims that the classical matrix in fact contains errors, but the 'correct' matrix provided by the authors is also slighly different than the one we generated ourselves. These differences could be imputable to the use of different clustering methods for each matrix (see previous point), so in a sense, all three matrices are correct since they all used the BLOCKS database with the algorithm of [HH92]. The problem is that this algorithm is not sufficiently well defined.

Best seqBLOSUM.

Our little experiment whose results are listed on figure 3.2.4 showed us that, for all BAliBASE datasets, the best matrix was always seqBLOSUM60.

Averaging seqBLOSUM and dynBLOSUM.
Small sum-of-pairs score improvements are possible when using the averaging method, and interestingly enough, the best averaging weights are always (60%, 40%) for seqBLOSUM62/dynBLOSUM62 and (50%, 50%) for seqBLOSUM30/dynBLOSUM30 (see figure 3.3.3). So it seems that the averaging weights stay the same for every dataset, although they contain different kinds of sequences.

Using DynaMine data is more useful for dissimilar sequences.
It seems that our averaging method works best for highly dissimilar sequences (see figure 3.3.9). This was expected: if very few residues are identical, it makes sense to use more structural data (such as DynaMine values) in a sequence alignment. However this experiment was done with a small dataset, so we should not jump to conclusions too quickly.

Of course, there is some valid criticism that one could make about our thesis. We think that the main issue is the narrow-minded focus: we used only one alignment algorithm (Needleman-Wunsch), one matrix-generating algorithm (BLOSUM), one benchmark database (BAliBASE), one scoring method (sum-of-pairs), one method of incorporating DynaMine values (averaging), we did not play with gap penalties, and we only worked with pairwise alignment (although multiple alignments seem to be more studied in bioinformatics). In particular, we are not convinced that BAliBASE was the best benchmark database for the job: since we wanted to use DynaMine values for improving alignments, a benchmark of highly dissimilar sequences would have been more interesting, because it is believed that structure is more conserved than sequences, and DynaMine gives us information on structure. But DynaMine is still a flexibility *predictor*, therefore, although it was built using benchmark experimental data, a large part of the information provided by DynaMine still comes from the amino acid sequence. Another point is the problem of the sum-of-pairs score for scoring everything, using another scoring system could have lead to different conclusions.

Appendix

This short appendix is meant to concisely explain where to obtain the C implementation of the Needleman-Wunsch-Gotoh algorithm developed for this thesis (see section 1.3), and how to use it. The source code is available on GitHub at `https://github.com/oboes/gotoh`. We tried to stay as minimalist as possible (around 500 lines of code in total) while still providing correct code. The implementation is C99-compliant, and no exotic features or libraries are used: in fact the only C header files needed for compiling are `stdio.h`, `stdlib.h`, `string.h`, and `ctype.h`. Therefore the code should be compilable on almost any C compiler.

The implementation is more of a small library, providing functions for creating a dynamic array (`gth_init`), setting substitution scores and gap penalties (`gth_set_sub` and `gth_set_gap`), filling and then backtracking the dynamic array (`gth_align`), and the freeing the allocated memory (`gth_free`). What distincts our implementation from other ones is that it allows a programmer to modify gap penalties and substitution scores depending on their positions in the sequences. Therefore the code can be used for much more than implementing a classical Needleman-Wunsch aligner: for example, it is possible to set gap penalties depending on the residues surrounding the gap or substitution scores which differ near the ends of a sequence. It is also possible to use the implementation for algorithms unrelated to biological sequences, such as Dynamic Time Warping (see section 1.3).

Basic input/output handling was also developed: the program can read FASTA files and NCBI/EMBOSS scoring matrix files, and a command-line interface is provided. We compared the results of our software, and got the exact same alignments as those produce by the `needle` program of the EMBOSS software package, thereforce we believe the implementation to be correct.

More information, as well as some C code example showcasing how to use the functions developed for this software is available on the GitHub repository of the program: `https://github.com/oboes/gotoh`.

Bibliography

[AGM+90] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myers, and David J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.

[BBC+91] Maria J. Berry, Laila Banu, Yoyi Chen, Susan J. Mandel, J. David Kieffer, John W. Harney, and P. Reed Larsen. Recognition of uga as a selenocysteine codon in type i deiodinase requires sequences in the 3' untranslated region. *Nature*, 353(6341):273–276, 1991.

[BD62] Richard E. Bellman and Stuart E. Dreyfus. Applied dynamic programming. Technical report, RAND Corporation, 1962.

[Bel52] Richard E. Bellman. On the theory of dynamic programming. *Proceedings of the National Academy of Sciences*, 38(8):716, 1952.

[Bel54] Richard E. Bellman. The theory of dynamic programming. Technical report, RAND Corporation, 1954.

[Bel84] Richard E. Bellman. *Eye of the Hurricane: an autobiography.* World Scientific, 1984.

[Ber06] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

[Bin06] Minou Bina, editor. *Gene Mapping, Discovery, and Expression.* Methods in Molecular Biology. Humana Press, 2006.

[Bre12] Eli Bressert. *SciPy and NumPy.* O'Reilly, 2012.

[BW07] Mark V Berjanskii and David S Wishart. The rci server: rapid and accurate calculation of protein flexibility using chemical shifts. *Nucleic acids research*, 35(suppl 2):W531–W537, 2007.

[BWF+00] Helen M. Berman, John Westbrook, Zukang Feng, Gary Gilliland, T. N. Bhat, Helge Weissig, Ilya N. Shindyalov, and Philip E. Bourne. The protein data bank. *Nucleic Acids Research*, 28(1):235–242, 2000. URL: `www.rcsb.org`.

[BWLH06] Gordon Blackshields, Iain M Wallace, Mark Larkin, and Desmond G Higgins. Analysis and comparison of benchmarks for multiple sequence alignment. *In silico biology*, 6(4):321–339, 2006.

[Con14] The UniProt Consortium. Activities at the Universal Protein Resource (UniProt), 2014. URL: `http://www.uniprot.org/`.

[CPT+13] Elisa Cilia, Rita Pancsa, Peter Tompa, Tom Lenaerts, and Wim F. Vranken. From protein sequence to dynamics and disorder with dynamine. *Nature Communications*, 4, 2013.

[CPT+14] Elisa Cilia, Rita Pancsa, Peter Tompa, Tom Lenaerts, and Wim F. Vranken. The dynamine webserver: predicting protein dynamics from sequence. *Nucleic Acids Research*, 2014.

[CR+13] Neil A. Campbell, Jane B. Reece, et al. *Biology.* Benjamin Cummings, 10th edition, 2013.

[DEKM98] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, 1998.

[DHH11] Celine Scornavacca Daniel H. Huson, Regula Rupp. *Phylogenetic Networks: Concepts, Algorithms and Applications.* Cambridge University Press, 2011.

[DLB+01] A Keith Dunker, J David Lawson, Celeste J Brown, Ryan M Williams, Pedro Romero, Jeong S Oh, Christopher J Oldfield, Andrew M Campen, Catherine M Ratliff, Kerry W Hipps, et al. Intrinsically disordered protein. *Journal of Molecular Graphics and Modelling*, 19(1):26–59, 2001.

[DS78] Margaret O Dayhoff and Robert M Schwartz. A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, 5(3):345–358, 1978.

[E+95] Sean R. Eddy et al. Multiple alignment using hidden markov models. In *Proceedings of the Intelligent Systems for Molecular Biology*, volume 3, pages 114–120, 1995.

[Edd04] Sean R. Eddy. Where did the blosum62 alignment score matrix come from? *Nature Biotechnology*, 22:1035–1036, 2004.

[Edg10] Robert C. Edgar. Quality measures for protein alignment benchmarks. *Nucleic acids research*, 38(7):2145–2153, 2010.

[Elo02] Arne Elofsson. A study on protein sequence alignment quality. *Proteins: Structure, Function, and Bioinformatics*, 46(3):330–339, 2002.

[Fit66] Walter M Fitch. An improved method of testing for evolutionary homology. *Journal of molecular biology*, 16(1):9–16, 1966.

[Fu04] Haian Fu, editor. *Protein-Protein interaction: Methods and Applications.* Methods in Molecular Biology. Humana Press, 2004.

[GML+10] M. Goujon, H. McWilliam, W. Li, F. Valentin, S. Squizzato, J. Paern, and R. Lopez. A new bioinformatics analysis tools framework at EMBL-EBI. *Nucleic Acids Research*, 738(539), 2010.

[Got82] Osamu Gotoh. An improved algorithm for matching biological sequences. *Journal of molecular biology*, 162(3):705–708, 1982.

[HGPH00] Jorja G. Henikoff, Elizabeth A. Greene, Shmuel Pietrokovski, and Steven Henikoff. Increased coverage of protein families with the BLOCKS database servers. *Nucleic acids research*, 28(1):228–230, 2000.

[HH91] Steven Henikoff and Jorja G. Henikoff. Automated assembly of protein blocks for database searching. *Nucleic Acids Research*, 19(23):6565–6572, 1991.

[HH92] Steven Henikoff and Jorja G. Henikoff. Amino acid substitution matrices from protein blocks. *Proc. Natl. Acad. Sci. USA*, 89(22):10915–10919, 1992.

[HHP99] Steven Henikoff, Jorja G. Henikoff, and Shmuel Pietrokovski. BLOCKS+: a non-redundant database of protein alignment blocks derived from multiple compilations. *Bioinformatics*, 15(6):471–479, 1999.

[Hir75] Daniel S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Communications of the ACM*, 18(6):341–343, 1975.

[IAE09] Kristoffer Illergård, David H. Ardell, and Arne Elofsson. Structure is three to ten times more conserved than sequence — a study of structural response in protein cores. *Proteins: Structure, Function, and Bioinformatics*, 77(3):499–508, 2009.

[IT00] Rieko Ishima and Dennis A Torchia. Protein dynamics from nmr. *Nature Structural and Molecular Biology*, 7(9):740–743, 2000.

[JB09] / PaweÅĆ Wojciechowski Jacek BÅĆaÅijewicz1, / Piotr Formanowicz1. Some remarks on evaluating the quality of the multiple sequence alignment based on the balibase benchmark. *International Journal of Applied Mathematics and Computer Science*, 19(4):675–678, 2009.

[Joh67] Stephen C Johnson. Hierarchical clustering schemes. *Psychometrika*, 32(3):241–254, 1967.

[JS13] Joël Janin and Michael JE Sternberg. Protein flexibility, not disorder, is intrinsic to molecular recognition. *F1000 biology reports*, 5, 2013.

[Kay98] Lewis E Kay. Protein dynamics from nmr. *Biochemistry and cell biology*, 76(2-3):145–152, 1998.

[KCN+03] Gregory V. Kryukov, Sergi Castellano, Sergey V. Novoselov, Alexey V. Lobanov, Omid Zehtab, Roderic Guigó, and Vadim N. Gladyshev. Characterization of mammalian selenoproteomes. *Science*, 300(5624):1439–1443, 2003.

[KR88] B.W. Kernighan and D. Ritchie. *C Programming Language*. Pearson Education, 1988.

[LP85] David J. Lipman and William R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227(4693):1435–1441, 1985.

[LP88] David J. Lipman and William R. Pearson. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85(8):2444–2448, 1988.

[LS82] Giovanni Lipari and Attila Szabo. Model-free approach to the interpretation of nuclear magnetic resonance relaxation in macromolecules. 1. theory and range of validity. *Journal of the American Chemical Society*, 104(17):4546–4559, 1982.

[LS02] Timo Lassmann and Erik LL Sonnhammer. Quality assessment of multiple alignment programs. *FEBS letters*, 529(1):126–130, 2002.

[LS05] Timo Lassmann and Erik L. L. Sonnhammer. Automatic assessment of alignment quality. *Nucleic Acids Research*, 33(22):7120–7128, 2005.

[MÖ7] Meinard Müller. Dynamic time warping. In *Information Retrieval for Music and Motion*, pages 69–84. Springer, 2007.

[Mad13] Thomas Madden. The blast sequence analysis tool. In *The NCBI Handbook*, chapter 16. National Center for Biotechnology Information, 2013. URL: `blast.ncbi.nlm.nih.gov`.

[Mit05] Ruslan Mitkov, editor. *Oxford Handbook of Computational Linguistics*. Oxford University Press, 2005.

[MM88] Eugene W. Myers and Webb Miller. Optimal alignments in linear space. *Computer applications in the biosciences: CABIOS*, 4(1):11–17, 1988.

[Mou08] David W. Mount. Using gaps and gap penalties to optimize pairwise sequence alignments. *Cold Spring Harbor Protocols*, 2008(6):pdb.top40, 2008.

[MUB+08] John L Markley, Eldon L Ulrich, Helen M Berman, Kim Henrick, Haruki Nakamura, and Hideo Akutsu. Biomagresbank (bmrb) as a partner in the worldwide protein data bank (wwpdb): new policies affecting biomolecular nmr depositions. *Journal of biomolecular NMR*, 40(3):153–155, 2008.

[NHH00] Pauline C. Ng, Jorja G. Henikoff, and Steven Henikoff. Phat: a transmembrane-specific substitution matrix. *Bioinformatics*, 16(9):760–766, 2000.

[NW70] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3):443–453, 1970.

[OA12] A.J.W. Orry and R. Abagyan. *Homology Modeling: Methods and Protocols*. Methods in Molecular Biology. Humana Press, 2012.

[Pát07] Miroslav Pátek. Branched-chain amino acids. In *Amino Acid Biosynthesis — Pathways, Regulation and Metabolic Engineering*, pages 129–162. Springer, 2007.

[PDS00] Andreas Prlić, Francisco S. Domingues, and Manfred J. Sippl. Structure-derived substitution matrices for alignment of distantly related sequences. *Protein Engineering*, 13(8):545–550, 2000.

[PKMH00] Cameron Platell, Sung-Eun Kong, Rosalie McCauley, and John C. Hall. Branched-chain amino acids. *Journal of gastroenterology and hepatology*, 15(7):706–717, 2000.

[RF02] C. Ramey and B. Fox. *Gnu Bash Reference Manual*. Network Theory, 2002.

[RLB00] Peter Rice, Ian Longden, and Alan Bleasby. EMBOSS: the European Molecular Biology Open Software Suite. *Trends in genetics*, 16(6):276–277, 2000.

[ROS+04] Predrag Radivojac, Zoran Obradovic, David K Smith, Guang Zhu, Slobodan Vucetic, Celeste J Brown, J David Lawson, and A Keith Dunker. Protein flexibility and intrinsic disorder. *Protein Science*, 13(1):71–80, 2004.

[RP02] J.T. Reese and William R. Pearson. Empirical determination of effective gap penalties for sequence comparison. *Bioinformatics*, 18(11):1500–1507, 2002.

[SC78] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 26(1):43–49, 1978.

[Sch08] Stewart Scherer. *A Short Guide to the Human Genome*. Cold Spring Harbor Laboratory Press, 2008.

[Sel74] Peter H. Sellers. On the theory and computation of evolutionary distances. *SIAM Journal on Applied Mathematics*, 26(4):787–793, 1974.

[SGK96] Shinji Sunada, Nobuhiro Go, and Patrice Koehl. Calculation of nuclear magnetic resonance order parameters in proteins by normal mode analysis. *The Journal of chemical physics*, 104(12):4768–4775, 1996.

[SJK02] Gayathri Srinivasan, Carey M. James, and Joseph A. Krzycki. Pyrrolysine encoded by uag in archaea: charging of a uag-decoding specialized trna. *Science*, 296(5572):1459–1462, 2002.

[SJRS08] Mark P. Styczynski, Kyle L. Jensen, Isidore Rigoutsos, and Gregory Stephanopoulos. BLOSUM62 miscalculations improve search performance. *Nature Biotechnology*, 26:274–275, 2008.

[SSHJ93] Mark P. Sawicki, Ghassan Samara, Michael Hurwitz, and Edward Passaro Jr. Human genome project. *The American journal of surgery*, 165(2):258–264, 1993.

[SST85] Fred Sherman, John W Stewart, and Susumu Tsunasawa. Methionine or not methionine at the beginning of a protein. *BioEssays*, 3(1):27–31, 1985.

[SWD+11] F. Sievers, A. Wilm, D.G. Dineen, T.J. Gibson, K. Karplus, W. Li, R. Lopez, H. McWilliam, M. Remmert, J. S oding, J.D. Thompson, and D. Higgins. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. *Molecular Systems Biology*, 7(539), 2011. URL: `www.ebi.ac.uk/Tools/msa/clustalo/`.

[SWF81] Temple F. Smith, Michael S. Waterman, and Walter M. Fitch. Comparative biosequence metrics. *Journal of Molecular Evolution*, 18(1):38–46, 1981.

[TGQS09] Paolo Tormene, Toni Giorgino, Silvana Quaglini, and Mario Stefanelli. Matching incomplete time series with dynamic time warping: an algorithm and an application to post-stroke rehabilitation. *Artificial Intelligence in Medicine*, 45(1):11–34, 2009.

[TKRP05] Julie D. Thompson, Patrice Koehl, Raymond Ripp, and Olivier Poch. BAliBASE 3.0: Latest developments of the multiple sequence alignment benchmark. *Proteins: Structure, Function, and Bioinformatics*, 61(1):127–136, 2005.

[Tom02] Peter Tompa. Intrinsically unstructured proteins. *Trends in biochemical sciences*, 27(10):527–533, 2002.

[TPP99a] Julie D. Thompson, Frédéric Plewniak, and Olivier Poch. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. *Bioinformatics*, 15(1):87–88, 1999.

[TPP99b] Julie D. Thompson, Frédéric Plewniak, and Olivier Poch. A comprehensive comparison of multiple sequence alignment programs. *Nucleic acids research*, 27(13):2682–2690, 1999.

[Vin68] T.K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

[VRD11] G. Van Rossum and F.L.J. Drake. *The Python Language Reference Manual*. Network Theory Limited, 2011.

[VWLW05] Ivo Van Walle, Ignace Lasters, and Lode Wyns. Sabmark: a benchmark for sequence alignment that covers the entire known fold space. *Bioinformatics*, 21(7):1267–1268, 2005.

[WB+13] James D. Watson, Tania A. Baker, et al. *Molecular Biology of the Gene*. Benjamin Cummings, 7th edition, 2013.

[XW+05] Rui Xu, Donald Wunsch, et al. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.