Abstract

With the emergence and growth of large databases of information, efficient methods for storage and processing are becoming increasingly important. The existence of a *metric* distance measure between data entities enables efficient index structures to be applied when storing the data. Unfortunately, this is often not the case. Amino acid substitution matrices, which are used to estimate similarities between proteins, do not yield metric distance measures. Finding efficient methods for converting a non-metric matrix into a metric one is therefore highly desirable. In this work, the problem of finding such conversions is approached by embedding the data contained in the non-metric matrix into a metric space. The embedding is optimized according to a quality measure which takes the original data into account, and a distance matrix is then derived using the metric distance function of the space.

More specifically, an evolutionary scheme is proposed for constructing such an embedding. The work shows how a coevolutionary algorithm can be used to find a spatial embedding and a metric distance function which try to preserve as much of the proximity structure of the non-metrix matrix as possible. The evolutionary scheme is compared to three existing embedding algorithms. Some modifications to the existing algorithms are proposed, with the purpose of handling the data in the non-metric matrix more efficiently. At a higher level, the strategy of deriving a metric distance function from a spatial embedding is compared to an existing algorithm which enforces metricity by manipulating the data in the non-metric matrix directly (the triangle fixing algorithm).

The methods presented and compared are *general* in the sense that they can be applied in any case where a non-metric matrix must be converted into a metric one, regardless of how the data in the non-metric matrix was originally derived.

The proposed methods are tested empirically on amino acid substitution matrices, and the derived metric matrices are used to search for similarity in a database of proteins. The results show that the embedding approach outperforms the triangle fixing approach when applied to matrices from the PAM family. Moreover, the evolutionary embedding algorithms perform best among the embedding algorithms. In the case of the PAM250 scoring matrix, a metric distance matrix is found which is more sensitive than the mPAM250 matrix presented in a recent paper. Possible advantages of choosing one method over another are shown to be unclear in the case of matrices from the BLOSUM family.

Preface

This thesis is submitted to the Norwegian University of Science and Technology in fulfillment of the requirements for the degree Master of Science. The work has been conducted at the Department of Computer and Information Science, from January 2005 to June 2005.

Acknowledgments

First of all I would like to thank my supervisors, Asc. Prof. Magnus Lie Hetland and Prof. Finn Drabløs, for tutoring me through the project. Their suggestions have been invaluable, and the steady stream of background literature they provided me with made it easy for me to get the project started. I would also like to thank Geir Kjetil Sandve, Petter Braute and Jorg Rødsjø for contributing with valuable inputs and ideas, and Vassilis Athitsos for answering questions regarding the BoostMap algorithm. A final thanks goes to friends and fellow MSc candidates for many a good time at Pia's Café after a long day in front of the computer.

Contents

List of Figures

Chapter 1

Introduction

With the emergence and growth of large databases of information, efficient methods for storage and processing are becoming increasingly important. In the field of bioinformatics, for example, large amounts of proteins are being encoded and stored as strings of symbols. When studying a specific protein, scientists need fast methods of finding proteins in existing databases which are similar to the one at hand. This enables them to gain knowledge of how proteins are related to each other in terms of evolutionary relationships. Similar search methods are required in other scenarios, for example when processing large databases of textual documents. In general, the problem of finding such methods are relevant in all cases where one is dealing with databases where inter-object distances or similarities are defined.

Because of the size and growth rate of current databases, and uninformed sequential scan of the database is not feasible in most cases. As with many other computational challenges, the demands of the problem can be eased by introducing a trade-off between speed and *sensitivity* (which, informally, can be described as the ability to identify all of the database objects which are similar in some sense to the one at hand). When processing protein databases, *heuristic* search algorithms are most commonly used. Such algorithms exploit the structures of the inter-object relationships to exclude from further consideration parts of the search space which are not likely to contain candidate solutions. Of course, this introduces a risk of excluding true positive hits from the solution, because the entire search space is not examined.

Instead of resorting to heuristic search methods, one might try to structure the data in a way which enables large subsets of objects to be excluded at an early stage of the search process. The central question is: Can the properties of the inter-object distances be used to identify such subsets? If one object from the subset can be excluded from further consideration, we would like the inter-object distance function to ensure that this holds for all the other objects in the subset as well. This way of structuring the data falls under the category of *indexing* methods. Such methods generally require the distance function to be *metric*. An introduction to metric spaces and distance functions will be given in chapter 2, but for now the following informal definition can be used:

- No inter-object distances can be negative.
- Given two objects X and Y, the distance from X to Y should be equal to the distance from Y to X.
- Given three objects X, Y and Z, the distance from X to Y via Z should not be shorter than the direct distance from X to Y.

Unfortunately, these conditions do not hold in many cases. In bioinformatics, the functions which describe the relationships between proteins are not metric. It is therefore highly desirable to find efficient methods for making the functions metric. The metric distance function should produce search results which are as similar as possible to search results obtained using the original distance function.

This thesis addresses the problem of converting a non-metric distance function into a metric one. Several methods and algorithms are proposed and compared. The following sections elaborate on the motivation and objectives of the work.

1.1 Motivation and problem definition

Although the methods proposed and investigated in this work are intended to be general to the problem of finding metric mappings for non-metric distance functions, the work has mainly been inspired by the specific problem of non-metricity in bioinformatics.

Functions which are used to compute the similarity between two proteins are specified as *substitution matrices*. As will be presented in depth in chapter 2, proteins can be represented as strings of symbols from a finite alphabet. The substitution matrices specify the costs of replacing one symbol by another, and are used to calculate the total similarity between two such strings.

There exists no straightforward way of converting such a substitution matrix into a metric distance matrix while still preserving the sensitivity in relation to database queries. Assuming that the sensitivity is governed by the information contained in the substitution matrix, the problem can be stated as: Find a method for generating a metric distance matrix which preserves as much of the information from the original matrix as possible.

As the methods are intended to be general, they should be based on processing the data in substitution matrices without regards to the methods originally used to derive the matrices.

1.2 Objectives

The work aims at proposing and comparing a number of different methods for generating a metric distance matrix from a non-metric one. The methods should be presented and evaluated both analytically and empirically. More specifically, the empirical analysis should be based on protein substitution matrices, and search results should be compared with search results obtained using the original matrices.

Finally, the work should give a conclusion as to whether the proposed methods and the generated metric distance matrices could be used to organize proteins in index structures. The design and implementation of such protein index structures will not be considered in the work.

1.3 Contributions

The work presents the idea of embedding the data of a non-metric substitution matrix into a metric space. A metric distance matrix can then be derived directly from this space. Furthermore, the work shows that the fitness of such metric embeddings can be evaluated by comparing ranked inter-object distances in the embedding with ranked inter-object similarities in the original matrix.

Several existing algorithms are suggested for the purpose of mapping elements into a metric space. The work compares these algorithms both analytically and empirically in terms of their ability to retain the information inherent in the input substitution matrix. In addition, the mapping algorithms are compared to an algorithm which enforces metricity of a non-metric matrix explicitly by manipulating the matrix values directly.

Two evolutionary embedding algorithms are proposed as possible solutions to the metric mapping problem. Their designs are inspired by possible limitations of existing algorithms.

The methods and algorithms presented are general and not tied to the case of protein substitution matrices. This means that they can be applied to any such matrix, regardless of how the data in it was originally calculated or derived.

1.4 Structure of the thesis

Chapter 2 presents background theory which is necessary to understand the problem at hand and how it can be solved. Since the methods are tested on protein substitution matrices, the material in this chapter will be biased towards proteins and protein databases. Chapter 3 presents the proposed methods and the concrete algorithms which are used to implement them. It suggests possible modifications to existing algorithms, and proposes two new evolutionary algorithms for solving the problem. Chapter 4 explains how experiments were carried out to evaluate the metric matrices produced by the algorithms. Experimental results are also presented in this chapter. Chapter 5 provides an analytical comparison of the methods and algorithms, a discussion of the experimental results and a synthesis of analytical an empirical results. Finally, chapter 6 recapitulates and concludes the thesis.

1.5 Notation

Some comments must be made regarding notations used in the remaining chapters of the thesis:

- Normal fonts are used when referring to algorithms and methods in their general and/or original form. Small capital letters are used when referring to an explicit specification in this report. For example, BoostMap refers to the BoostMap algorithm in its general and original form, while BOOSTMAP refers to the specific pseudocode presented in section 3.5. Some small procedures referred to in small capital letters inside algorithms are explained in words in the text rather than in pseudocode.

- In algorithms, blocks of statements are specified by indentation. As an example, consider the following lines of pseudocode:

  ```
  1: if condition met then
  2:     Statement 1
  3:     Statement 2
  4: Statement 3
  ```

  Because statements 1 and 2 are indented, they both belong to the body of the **if** test, and are therefore only run if the condition is met. Statement 3, in contrast, is run regardless of whether the condition is met or not.

- A notation where numerous variable assignments can occur on a single line in an algorithm is used. The ordering of symbols on either side of the assignment operator determines the assignments. For example, line 1 below is equivalent to lines 2 and 3.

  ```
  1: $X, Y \leftarrow A, B$
  2: $X \leftarrow A$
  3: $Y \leftarrow B$
  ```

Chapter 2

Background

This chapter presents background theory which is necessary in order to understand the context of the problem and the advantages of metric distance functions. The sections are organized in a natural order, where transitions are as smooth as possible. Section 2.1 gives an overview of how strings of symbols are used to represent genetic sequences and proteins. In section 2.2, methods for computing the similarity between two such strings of symbols are introduced. The connection to distance functions is made in section 2.3, which introduces and presents amino acid substitution matrices. Section 2.4 defines and describes metric spaces and distance functions, while 2.5 shows how the properties of such spaces can be used to organize data in efficient index structures. This latter section also discusses some of the challenges of non-metric distance functions and what characteristics a mapping from a non-metric to a metric distance function should have. Finally, section 2.6 reviews some recent related works.

2.1 Sequences in computational biology

As mentioned, the methods proposed and analyzed in this work for converting non-metric distance functions to metric ones have been tested in the field of bioinformatics. More specifically, amino acid substitution matrices are subjected to the proposed methods, and the resulting distance functions are tested by searching in a database of proteins. Thus, many methods and concepts in this thesis will be illustrated using examples from bioinformatics, which in turn requires a brief introduction to some important concepts to be given. The introduction is concerned with data processing in the field and computational challenges which lie therein. A more thorough introduction to the background of computational biology can be found in a book like [Wat95].

2.1.1 Genetic sequences

A genetic sequence, sometimes also called a *DNA* sequence, is a string of symbols from an alphabet consisting of the four nucleotide bases of DNA - adenine, cytosine, guanine and thymine. The symbols A, C, G and T are commonly used to represent these nucleotide bases. They encode the *chromosomes* of an organism, which contain genes and other nucleotide subsequences. A complete DNA sequence of the chromosomes of an organism is called the *genome* of the organism. The Human Genome Project[1], started in 1986, aimed at mapping the human genome at nucleotide level. Another goal was to identify all genes present in the human genome. The project was completed two years ahead of schedule in april 2003.

The *genetic code* is a set of rules for mapping DNA sequences (genes) to proteins. It determines the structure of the proteins in a process called protein synthesis. Proteins are composed of *amino acids*, and a combination of three nucleotide bases in a genetic sequence codes a single amino acid. Although there are a total of 64 possible combinations of three nucleotide bases, only 61 are used in the synthesis. In addition, several different triples may code the same amino acid.

2.1.2 Proteins - sequences of amino acids

Proteins are molecular structures in the form of chains of amino acids. A total of 20 amino acids are encoded by the genetic code. These are called proteinogenic or standard amino acids. Thus, a protein may be described as a string of symbols from an alphabet of 20 symbols. The amino acids, together with the symbols commonly used to represent them, are shown in table 2.1.

Table 2.1: The 20 amino acids

Abbr.	**Name**	**Abbr.**	**Name**
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	Treonine
G	Glycine	W	Tryptophan
H	Histidine	Y	Tyrosine
I	Isoleucine	V	Valine

[1]http://gdbwww.gdb.org/

```
MRFLRGFVFSLAFTLYKVTATAEIGSEINVENEAPPDGLSWEEWHMDHEHQLKDYTPET
FFALHDIKKKGFLDENDILSLYGLNREEIVGAGDGMGQHDESEKIDNEMAKRVVSLIMR
LLDVDDNTKITKEEYLQFAKRGNKFPDLGVGVGHHSDFELEYEIHHWNKFHKDKDPDVK
VVHKEDIEHELLHHEHEIEHEEEIQRGASRATVITDDELESRIELKNIPEKFKNGIF
```

Figure 2.1: Secretory protein (SSp120p) in Saccharomyces cerevisiae

An example of a protein represented as a string of amino acid symbols is shown in figure 2.1. The protein shown is the secretory protein in the yeast saccharomyces cerevisiae, which is commonly known as baker's or budding yeast.

Since the human genome project was completed in 2003, focus has shifted towards finding functional descriptions of the more than 30.000 proteins encoded by the human genome. This is commonly referred to as the *human proteome project*. So far, only a small fraction of the proteins has been mapped. Since almost everything in the body involves or is made from proteins, it is very desirable to gain knowledge of how they function and relate to each other. Such an understanding could be the first step towards finding cures for diseases which involve mapped proteins.

The amount of data available, and the computational demands of the methods used to compare different proteins, makes this a very challenging task. Much research activity has therefore recently focused on finding more efficient ways of handling the data at hand. The existence of a metric function specifying the distances between different proteins would enable us to store the data in efficient index structures. Metric spaces and distance functions are presented in section 2.4, and an overview of data indexing based on the properties of such spaces is given in section 2.5.

2.2 Computing similarity between symbol sequences

Many problems involve the calculation of similarity between strings of symbols. Examples of such problems include searching for similar patterns in time series, finding replacement words in a spelling corrector, and identifying similar biological sequences like the ones presented in the previous section. Searching for similar patterns in time series can include any kind of string, for example a sequence of discrete signals sampled at a uniform rate. A spelling corrector would obviously work on strings of characters from an alphabet, such as the latin one. Identifying similar biological sequences involves analysis of strings from an alphabet consisting of biological base units. Table 2.1 presents such an alphabet, where each amino acid is associated with a symbol.

This section introduces some important concepts and algorithms necessary to understand how such sequence similarity may be calculated. Subsection 2.2.1 introduces some important concepts and terminology used when describing such

```
TTGACACCCTCC-CAATTGTA
    ::  ::   ::  :
ACCCCAGGCTTTACACAT---
```

Figure 2.2: Example of a global alignment

calculations. The remaining subsections describe how the calculations may be done.

2.2.1 Terminology and concepts

The process of searching for similarity between strings of symbols is often described as an *alignment* of the two sequences. The actual result of the calculation is called the *score* of the alignment. There are two main classes of alignments: *global* alignments and *local* alignments.

Global alignments treat the two input sequences as potentially equivalent. The alignment process aims to identify conserved regions and regions of difference in the sequences. An alignment can be illustrated by writing the sequences on separate lines and marking the matching symbols. Figure 2.2 shows an example, where the global alignment between the two genetic sequences TTGACACCCTCCCAATTGTA and ACCCCAGGCTTTACACAT is illustrated (adapted from [GD91]).

An example application of global alignments is the comparison of genes or proteins with similar functionality. Well known algorithms for computing global alignments include the simple edit distance algorithm presented in section 2.2.2, and the Needleman-Wunsch algorithm introduced in [NW70]. Since this report is mainly concerned with local alignments (to be explained below), the latter algorithm is not presented here. The simple edit distance algorithm is presented as a general example of how dynamic programming can be used to estimate the similarity between two sequences.

In contrast to global alignments, local alignments aim to find the two best matching *subsequences*. An example of such an alignment is shown in figure 2.3. Local alignments are used when searching for conserved regions in two proteins, or when searching for local similarities in large sequences in general. A well known algorithm for computing the local alignment of two sequences is the Smith-Waterman dynamic programming algorithm. This algorithm is presented in section 2.2.3.

The process of computing alignments between sequences is often called *homology search*. Such searches are inherently computationally expensive, since the computation of alignments is an order of magnitude more complex than a linear scan to check for an exact match between sequences. Alignment algorithms employ *scored matching*, where different scores are given for the preservation of

```
---------TTGACACCCTCCCAATTGTA      TTGACAC
         :: ::::                or :: ::::
ACCCCAGGCTTTACACAT-----------      TTTACAC
```

Figure 2.3: Example of a local alignment

different symbols in the alphabet in consideration, and penalties are given for gaps in the alignment. The resulting alignment is the one with the highest score.

Because of the computational complexity, finding the best matches between a query sequence and a large database by computing the alignment with all of the stored sequences is often not feasible in practice. Therefore, a number of commonly used heuristic alignment algorithms have been developed. Although this work is not concerned with heuristic high-performance alignment algorithms, brief descriptions of some of these are given in section 2.2.4.

2.2.2 Simple edit distance

One of the simplest measures of similarity between two sequences of symbols is the *edit distance*. Given the costs of deleting a symbol, inserting a symbol and replacing a symbol by another, this distance represents the minimal cost of transforming one sequence into the other. Such a simple scheme is rarely used in practice, but it serves as a basic example of how dynamic programming is usually used in most alignment algorithms.

Three cost constants are given; *del* represents the cost of deleting a symbol, *ins* represents the cost of inserting a symbol and *sub* represents the cost of replacing a symbol with another symbol. Usually, *sub* is chosen to be less than the sum of *del* and *ins*. Representing the edit distance from the first i symbols of sequence s to the first j symbols of sequence t by $d_{i,j}$, equations 2.1 to 2.4 can be used to calculate the total distance between the two sequences.

$$d_{0,0} = 0 \tag{2.1}$$

$$d_{i,0} = d_{i-1,0} + del \tag{2.2}$$

$$d_{0,j} = d_{0,j-1} + ins \tag{2.3}$$

$$d_{i,j} = \min \begin{cases} d_{i-1,i} + del \\ d_{i,j-1} + ins \\ \begin{cases} d_{i-1,j-1} & s_i = t_j \\ d_{i-1,j-1} + sub & s_i \neq t_j \end{cases} \end{cases} \tag{2.4}$$

Equation 2.2 initializes the distances from the first i symbols of s to an empty sequence. For example, $i = 3$ gives the distance from the first three symbols of s to an empty sequence, which corresponds to three deletions. Equation 2.3

		S	Y	M	B	O
	0	1	2	3	4	5
S	1	0	1	2	3	4
Y	2	1	0	1	2	3
S	3	2	1	2	3	4
T	4	3	2	3	4	5
O	5	4	3	4	5	4

Figure 2.4: Edit distance between two character strings

initializes the distances from an empty string to the first j symbols of t. This corresponds to j insertions. Equation 2.4 gives the general recurrence relationship from which all other distances are calculated. This relationship states that the cost of transforming $[s_1, s_i]$ to $[t_1, t_j]$ can be seen as the minimum cost of transforming

1. $[s_1, s_{i-1}]$ to $[t_1, t_j]$ and deleting a symbol from t.
2. $[s_1, s_i]$ to $[t_1, t_{j-1}]$ and inserting a symbol into s.
3. $[s_1, s_{i-1}]$ to $[t_1, t_{j-1}]$, if s_i and t_j are equal.
4. $[s_1, s_{i-1}]$ to $[t_1, t_{j-1}]$ and replacing s_i by t_j, if these two symbols are unequal.

It is quite clear that this recursive relationship has both overlapping subproblems and optimal substructure, and that dynamic programming can therefore be applied to solve it. The memoization process can be seen as the process of filling in the values of a two-dimensional array where the rows correspond to the symbols of s and the columns correspond to the symbols of t. An example of such an array is given in figure 2.4. For each cell in the array, three neighbouring cells must be examined to determine its value. Therefore, it is obvious that the complexity of the algorithm is $\Theta(mn)$, where m is the length of s and n is the length of t. The total edit distance between the two sequences is found as $d_{m,n}$.

2.2.3 The Smith-Waterman local alignment algorithm

A common way of identifying common *subsequences* in two strings of symbols is the Smith-Waterman algorithm introduced in [SW81]. This algorithm utilizes dynamic programming to find the pair of subsequences, one from each string, with the highest degree of homology.

To describe the algorithm, let $S = [s_1, s_2, ..., s_n]$ denote the first sequence and $T = [t_1, t_2, ..., t_m]$ the second one. A substitution matrix $S_{i,j}$ is assumed to be known. This matrix gives a measure of similarity for each pair of symbols in the alphabet. For example, in the case of homology searching in proteins S would be a 20×20 matrix. A similarity matrix H of size $m \times n$ is set up. An element (i, j) of this matrix represents the highest similarity observed between two subsequences ending in s_i and t_j. The matrix is initialized by setting

$$H_{k0} = H_{0l} = 0 \text{ for } 0 \leq k \leq n \text{ and } 0 \leq l \leq m \tag{2.5}$$

Thereafter, the remaining matrix elements ($1 \leq i \leq n$ and $1 \leq j \leq m$) are calculated using dynamic programming on the recurrence relationship

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + S_{s_i,t_j}, \\ \max\limits_{l \geq i}\{H_{i,j-l} - W_l\}, \\ \max\limits_{k \geq j}\{H_{i-k,j} - W_k\}, \\ 0 \end{cases} \tag{2.6}$$

where W_k represents the cost of a deletion of length k. As can be seen, there are four possibilities at each position (i, j) of the similarity matrix. Either s_i and t_j are associated (and thus part of the subsequence of highest similarity observed so far), s_i is at the end of a deletion of length k, t_j is at the end of a deletion of length l, or no similarity has been observed so far in this subsequence. The last case (zero similarity observed so far) is enforced because we do not allow negative similarity values for subsequences.

The straightforward Smith-Waterman algorithm is clearly $O(n^2m)$ if $n > m$ or $O(m^2n)$ if $n < m$. However, [Got82] introduces some modifications to the original algorithm which have the effect of reducing the computational complexity to $O(nm)$. In his work, Gotoh introduces two addition matrices P and Q. $P_{i,j}$ holds the score of the highest scoring alignment between $[s_1, s_i]$ and $[t_1, t_j]$, given that the alignment ends with s_i aligned to an insertion or a deletion. Similarly, $P_{i,j}$ holds the score of the highest scoring alignment between $[s_1, s_i]$ and $[t_1, t_j]$, given that the alignment ends with an insertion or a deletion aligned to t_j. The recursive relation assumes that we are given an *affine* penalty model, i.e. a model which can be written as

$$W_k = -o - (k-1)e \tag{2.7}$$

where k is the length of the gap, o is the penalty for opening a new gap and e is the penalty for extending an existing gap. Using this affine gap model, Gotoh shows that the recurrence relationship can be rewritten as

$$H_{i,j} = \max \begin{cases} H_{i-1,j-1} + S_{s_i,t_j}, \\ P_{i-i,j-i} + S_{s_i,t_j}, \\ Q_{i-1,j-1} + S_{s_i,t_j} \end{cases} \tag{2.8}$$

$$P_{i,j} = \max\{H_{i-l,j} - o, P_{i-1,j} - e\} \tag{2.9}$$

$$Q_{i,j} = \max\{H_{i,j-1} - o, Q_{i,j-1} - e\} \tag{2.10}$$

The derivations of $P_{i,j}$ and $Q_{i,j}$ are based on the original recurrence relationship in equation 2.6. If $P_{i,j}$ is chosen to represent $\max_{1\leq k\leq i}\{H_{i-k,j} - o - (k-1)e\}$, it can be rewritten as

$$\begin{aligned} P_{i,j} &= \max_{1\leq k\leq i}\{H_{i-k,j} - o - (k-1)e\} \\ &= \max\{H_{i-1,j} - o, \max_{2\leq k\leq i}\{H_{i-k,j} - o - (k-1)e\}\} \\ &= \max\{H_{i-1,j} - o, \max_{1\leq k\leq i-1}\{H_{i-1-k,j} - o - ke\}\} \\ &= \max\{H_{i-1,j} - o, P_{i-1,j} - e\} \end{aligned} \tag{2.11}$$

$Q_{i,j}$ can be rewritten in the same way. Thus, the cost of the most similar subsequences can be computed in $O(nm)$ time if $W_k = -o - (k-1)e$, by storing the $P_{i,j}$ and $Q_{i,j}$ values along the way when computing the values of the similarity matrix H. The final score is found as the maximum local alignment score observed while iterating through the algorithm.

2.2.4 Searching in large databases

In practice, aligning a query protein with all proteins in a database using the Smith-Waterman algorithm is not feasible. Many heuristic search algorithms have been developed for easing the computational demands of such linear database scans. Among these, BLAST and FASTA are by far the most commonly used ones. As such heuristic search algorithms are not directly linked to this work, none of them will be presented in detail here. However, brief descriptions of the two mentioned algorithms are included as examples of methods which are commonly used at the present time.

BLAST (Basic Local Alignment Search Tool) was introduced by Altschul et al. in [AGM$^+$90]. It works by breaking the query sequence and all database sequences into fragments of a specified length. The query fragments are then matched with the database fragments. The pairs which score lower than a certain threshold are rejected. For the ones which are selected for further investigation, the alignments are extended in either direction in an attempt to construct a full alignment.

FASTA was introduced by Lipman and Pearson in [LP85]. As BLAST, FASTA works by attempting to match fragments of the query sequence and database sequences. The difference is that FASTA identifies exact matches. The 10 regions with the highest density of matches in a protein are then selected and re-scored using an appropriate substitution matrix. When the scores of regions exceed a certain threshold, the regions are joined to form a larger region with gaps. Finally, a variation of the Smith-Waterman algorithm is used to re-score the joined region.

2.3 Substitution matrices

As explained in the previous section, algorithms for computing alignments use some kind of scoring scheme to compute the cost of an alignment. Simple edit distance only considers penalties, whereas more sophisticated methods give scores to matching pairs of symbols in the two sequences to be aligned. The Smith-Waterman algorithm introduced in section 2.2.3, for example, requires the scores of all possible pairs from the alphabet of symbols to be specified. Such scores may be specified in the form of a matrix where the cells correspond to the different pairs of symbols. Scoring matrices are often called *substitution matrices* because a cell specifies the score or penalty obtained when substituting one symbol for another. This section gives an introduction to such substitution matrices. Subsection 2.3.1 provides some examples of basic substitution matrices. Subsection 2.3.2 gives an overview of substitution matrices used in bioinformatics when searching for similarity in proteins. Finally, 2.3.3 points out some issues which occur when attempting to convert a similarity matrix into a dissimilarity matrix.

2.3.1 Some simple substitution matrices

The most simple substitution matrix imaginable is the identity matrix. An example is shown in figure 2.5, where the alphabet consists of the four nucleotide bases of DNA. This scheme awards identical nucleobases in the same position in an alignment with a score of 1.0. Dissimilar nucleobases in the same position receives no award, but no penalty either. The same scheme can be defined for protein scoring by using the 20×20 identity matrix corresponding to the 400 possible pairs of amino acids.

Two other simple substitution matrices are shown in figure 2.6. Figure 2.6a shows a matrix where identical nucleobases in the same position in an alignment are awarded a score of 5, whereas dissimilar nucleobases in the same position are penalized with a score of -4. Both this matrix and the identity matrix in figure 2.5 are examples of scoring schemes based on similarity; symbols in the same position in an alignment receive scores according to their degree of

	A	T	C	G
A	1	0	0	0
T	0	1	0	0
C	0	0	1	0
G	0	0	0	1

Figure 2.5: Identity score matrix

conformity. In contrast, figure 2.6b shows an example of a *dissimilarity* based substitution scheme. The entries of the matrix correspond to distances between symbols rather than similarity values, so that symbols defined to be similar are close and dissimilar ones are farther apart. This means that we are interested in finding alignments of low dissimilarity instead of high similarity, and the algorithms used when dealing with similarity scores cannot be used unmodified. Such issues are covered in subsection 2.3.3.

a)

	A	T	C	G
A	5	-4	-4	-4
T	-4	5	-4	-4
C	-4	-4	5	-4
G	-4	-4	-4	5

b)

	A	T	C	G
A	0	5	5	1
T	5	0	1	5
C	5	1	0	5
G	1	5	5	0

Figure 2.6: a) Similarity matrix b) Dissimilarity matrix

2.3.2 Amino acid substitution matrices

A number of commonly used substitution matrices exist for searching in protein databases. Amino acid substitution matrices contain 400 entries, each giving a measure of the similarity or dissimilarity between a pair of amino acids. By far the most commonly used ones are the matrices from the PAM and BLOSUM families.

PAM (Point Accepted Mutation) matrices were introduced by Dayhoff et al. in [DSO78]. Each PAM matrix is associated with a number which specifies the expected number of mutations per 100 amino acids. For example, the PAM250 matrix applies to time intervals of 250 mutations per 100 amino acids. In other words, the number associated with a PAM matrix specifies the degree of evolutionary divergence which it applies to. The PAM250 matrix is included as a general example of an amino acid substitution matrix in figure 2.7.

The BLOSUM (BLOcks SUbsitution Matrices) family of matrices was introduced in [HH92]. These matrices are derived using another method and another dataset than the PAM model uses. BLOSUM matrices are also associated with numbers. Generally speaking, BLOSUM matrix numbers approximately match the percentage of identical amino acids in the aligned sequences (i.e. BLOSUM62 for sequences which are 50% identical).

	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**2**	-2	0	0	-2	0	0	1	-1	-1	-2	-1	-1	-3	1	1	1	-6	-3	0
R	-2	**6**	0	-1	-4	1	-1	-3	2	-2	-3	3	0	-4	0	0	-1	2	-4	-2
N	0	0	**2**	2	-4	1	1	0	2	-2	-3	1	-2	-3	0	1	0	-4	-2	-2
D	0	-1	2	**4**	-5	2	3	1	1	-2	-4	0	-3	-6	-1	0	0	-7	-4	-2
C	-2	-4	-4	-5	**12**	-5	-5	-3	-3	-2	-6	-5	-5	-4	-3	0	-2	-8	0	-2
Q	0	1	1	2	-5	**4**	2	-1	3	-2	-2	1	-1	-5	0	-1	-1	-5	-4	-2
E	0	-1	1	3	-5	2	**4**	0	1	-2	-3	0	-2	-5	-1	0	0	-7	-4	-2
G	1	-3	0	1	-3	-1	0	**5**	-2	-3	-4	-2	-3	-5	0	1	0	-7	-5	-1
H	-1	2	2	1	-3	3	1	-2	**6**	-2	-2	0	-2	-2	0	-1	-1	-3	0	-2
I	-1	-2	-2	-2	-2	-2	-2	-3	-2	**5**	2	-2	2	1	-2	-1	0	-5	-1	4
L	-2	-3	-3	-4	-6	-2	-3	-4	-2	2	**6**	-3	4	2	-3	-3	-2	-2	-1	2
K	-1	3	1	0	-5	1	0	-2	0	-2	-3	**5**	0	-5	-1	0	0	-3	-4	-2
M	-1	0	-2	-3	-5	-1	-2	-3	-2	2	4	0	**6**	0	-2	-2	-1	-4	-2	2
F	-3	-4	-3	-6	-4	-5	-5	-5	-2	1	2	-5	0	**9**	-5	-3	-3	0	7	-1
P	1	0	0	-1	-3	0	-1	0	0	-2	-3	-1	-2	-5	**6**	1	0	-6	-5	-1
S	1	0	1	0	0	-1	0	1	-1	-1	-3	0	-2	-3	1	**2**	1	-2	-3	-1
T	1	-1	0	0	-2	-1	0	0	-1	0	-2	0	-1	-3	0	1	**3**	-5	-3	0
W	-6	2	-4	-7	-8	-5	-7	-7	-3	-5	-2	-3	-4	0	-6	-2	-5	**17**	0	-6
Y	-3	-4	-2	-4	0	-4	-4	-5	0	-1	-1	-4	-2	7	-5	-3	-3	0	**10**	-2
V	0	-2	-2	-2	-2	-2	-2	-1	-2	4	2	-2	2	-1	-1	-1	0	-6	-2	**4**

Figure 2.7: PAM250 score matrix

The details of how PAM and BLOSUM matrices are derived are not necessary to understand the algorithms and methods presented in this work. Nevertheless, detailed descriptions of the two models are included in appendix A for readers who want a basic understanding of the meaning of the data contained in the matrices. The appendix also contains a brief survey of other commonly used amino acid substitution matrices.

2.3.3 Similarity versus dissimilarity

In many cases, it is desirable to be able to convert similarity matrices like the ones presented in the previous section into *dissimilarity* (distance) matrices. As will be shown in the next section, metric spaces are based on the properties of the distance function associated with the them. The first step towards the construction of a metric matrix from a substitution matrix will therefore often be such a conversion.

If the diagonal elements of the similarity matrix are equal, the conversion simply involves subtracting each matrix cell value from this diagonal value to obtain a new cell value. Unfortunately, this is not the case with most amino acid substitution matrices. We would like the conversion to produce a (possibly non-metric) distance matrix with zeros on the diagonal. However, since the diagonal elements in the similarity matrix are not equal, we cannot simply subtract a value from each matrix cell.

A conversion method introduced by Linial et al. in [LLTY97] and later generalized by Halpering et al. in [HBK$^+$03] is shown in equation 2.12, where D is the distance matrix and S is the similarity matrix.

$$D_{u,v} = S_{u,u} + S_{v,v} - 2S_{u,v} \tag{2.12}$$

Originally proposed as a distance measure for subsequences of fixed length, Halperin et. al extend the definition to subsequences of arbitrary length, and use the equation 2.12 to produce distance values from similarity scores. The equation is also adopted in this work in cases where such a conversion is needed. It is questionable whether the method is able to preserve all information from the similarity matrix. This will be elaborated on in section 3.5.3. As will become apparent in chapter 3, however, some of the proposed algorithms use the step only as an initial estimate – the final metric matrix is produced using data from the original similarity matrix.

2.4 Metric spaces and distance functions

The computational demands of many problems, including the search for similarity in a database of proteins, can be eased considerably when the similarity or dissimilarity measure to be used possesses the properties of a *metric* system. Unfortunately, this is often not the case. As the concern of this work is to propose and compare methods for deriving distance metrics from non-metric similarity or dissimilarity measures, this section introduces the concept of metric spaces and distance functions. Section 2.4.1 provides the necessary formal definitions, while section 2.4.2 gives some examples of metric and non-metric distance functions. Finally, section 2.4.3 discusses how *metricity* can be measured.

2.4.1 Formal definitions

Assume that we have a set S of elements and a function d which for all ordered pairs of elements (a, b) from S returns the distance $d(a, b)$ from a to b. Some desirable properties of the function d are defined in equations 2.13 to 2.16.

$$\forall a, b \in S : d(a, b) = d(b, a) \tag{2.13}$$

$$\forall a, b \in S : d(a, b) = 0 \Leftrightarrow a = b \tag{2.14}$$

$$\forall a, b \in S : d(a, b) \geq 0 \tag{2.15}$$

$$\forall a, b, c \in S : d(a, b) \leq d(a, c) + d(c, b) \tag{2.16}$$

Equation 2.13 requires that *symmetry* is satisfied for all distances between pairs of elements in S. Equation 2.14 requires all distances from elements of S to themselves to be of zero length. Equation 2.15 states that all distances returned by the function d should be non-negative. The *triangle inequality* is satisfied for function d if condition 2.16 holds.

From these four conditions, some classes of spaces can be defined. First, define a *space* to be a pair (S, d) consisting of a set S of elements and a distance function

$d : S \times S \to \mathbb{R}$. A space where the distance function satisfies conditions 2.13 and 2.14 is called a *premetric* space. A premetric space whose distance function also satisfies condition 2.15 is called a *semimetric* space. *Metric* spaces are spaces where the distance functions satisfy all four conditions. If the triangle inequality is strengthened to $d(a,b) \leq \max\{d(a,c), d(c,b)\}$ for all $a,b,c \in S$, the space is called *ultrametric*. Since the distance function determines if a space is metric or not, one often talks about metric or non-metric distance functions instead of spaces.

From a computational point of view, the properties of metric spaces are most desirable. This will become apparent in section 2.5, where the concepts of data indexing are introduced.

2.4.2 Some metric and non-metric distance functions

Many distance functions satisfy the conditions of metricity. The most common metric distance functions are contained in the set of L_p norms, where $p \geq 1$ is required for metricity. These distance functions are defined as shown in equation 2.17, where x_{ak} is the value of element a along the kth axis of the D dimensions in which the elements are embedded.

$$d_p(a,b) = \left(\sum_{k=1}^{D} |x_{ak} - x_{bk}|^p \right)^{1/p} \tag{2.17}$$

The most commonly used L_p norms are given by $p \in \{1, 2, \infty\}$. They are shown in equations 2.18, 2.19 and 2.20. Setting $p = 1$ gives a distance function which is often referred to as the *Manhattan* or *city block* distance. The equation reduces to the *euclidean* distance function when setting $p = 2$.

$$d_1(a,b) = \sum_{k=1}^{D} |x_{ak} - x_{bk}| \tag{2.18}$$

$$d_2(a,b) = \sqrt{\sum_{k=1}^{D} (x_{ak} - x_{bk})^2} \tag{2.19}$$

$$d_\infty(a,b) = \max_{1 \leq k \leq D} |x_{ak} - x_{bk}| \tag{2.20}$$

Many physical systems have distance functions which are not metric. For example, consider a distance function which specifies the travel time distance between cities. For a set of four cities, such a distance function could be represented as the distance matrix shown in figure 2.8. This function does not satisfy the triangle inequality, because $d(C,D) > d(C,B) + d(B,D)$. Also note that

	A	B	C	D
A	0	3	2	5
B	3	0	1	4
C	2	1	0	6
D	5	4	6	0

Figure 2.8: An example of a distance matrix

although this distance matrix is symmetric, this is not necessarily the case for distance measures such as travel time distance. We could perfectly well have a non-symmetric distance matrix representing travel time distance (e.g. the routes could include unidirectional paths).

Another example of non-metric distance functions is the set of mutation probability matrices used when searching for similarity between protein sequences, including the PAM and BLOSUM family of matrices. Although these matrices contain *similarity scores* instead of distances, they are non-metric in the sense that there exists no straightforward ways of converting the similarity scores to metric distance functions without losing information.

2.4.3 Measuring metricity

When evaluating a distance function, it is often desirable to measure its *metricity*. An exact formal definition of what is meant by metricity is hard to specify. Intuitively, one would like the metricity of a matrix to represent how close the matrix is to being metric.

Thus, there are several possible ways of defining the metricity. Assume that the distance function is given in the form of a distance matrix, and that the distance matrix satisfies the conditions given in 2.13, 2.14 and 2.15. The perhaps most straightforward measure of metricity would be a simple count of the number of triangle inequality violations. This measure does not, however, take into account *how much* the triangle inequalities are violated. The amount of violation can be seen as the difference between $d(a,c) + d(c,b)$ and $d(a,b)$ for a triangle (a,b,c). Intuitively, one would like a matrix where triangle inequalities are violated by a small amount to have a higher degree of metricity than matrices where the same triangle inequalities are violated by larger amounts.

One possible measure of metricity can be derived from [CNBYM99], where a relaxation of the triangle inequality is suggested to enable indexing of data based on non-metric distance functions. Assume that we can find a constant α such that the triangle inequality shown in equation 2.21 holds for all triples (a,b,c) of elements. Then α could be adapted as a measure of metricity.

$$d(a,b) \leq \alpha[d(a,c) + d(c,b)] \tag{2.21}$$

The problem with using α as metricity measure is that the constant must be chosen high enough to relax the worst triangle inequality violation, and does not take into account how much the other triangle inequalities are violated. The metricity measure introduced in [GKM03] therefore seems like a better choice. This measure can be calculated as $1 - nm$, where nm is calculated as given by equation 2.23. The value nm is a real number in the interval $[0, 1]$ measuring the *non-metricity* of the distance function, i.e. $nm = 0$ for a metric matrix.

$$d'(a,b) = \min \begin{cases} d(a,b), \\ \min\limits_{c \neq a,b} \{d(a,c) + d(c,b)\} \end{cases} \tag{2.22}$$

$$nm = \text{average}_{a,b} \left(\frac{d(a,b) - d'(a,b)}{d(a,b)} \right) \tag{2.23}$$

If the distance matrix is metric, equation 2.22 gives $d'(a,b) = d(a,b)$ for all ordered pairs (a,b). But if the matrix is non-metric, one would find that for some of the distances in the matrix there exists a non-empty set $M \in \mathbb{N}$ such that $\forall c\{c \in M \rightarrow d(a,c) + d(c,b) < d(a,b)\}$, i.e. the triangle inequality is violated for the triangle (a,b,c). For these distances, equation 2.22 gives $d'(a,b) = \min_{c \neq a,b}\{d(a,c) + d(c,b)\}$, which increases the value of nm.

2.5 Data indexing

As mentioned in the previous section, the properties of metric spaces can be used to efficiently index the data. If the distance function used to calculate the similarity or dissimilarity between entities of the database satisfies the triangle inequality, a number of well known indexing methods can be used. This enables queries to be performed in sublinear time.

Section 2.2.4 provided an overview of common heuristics used to ease the computational demands of a linear database scan. The problem with such heuristic methods, however, is that the results returned in response to a query are not necessarily correct. In contrast, queries on an index always produce correct results if the data to be organized possesses the properties assumed by the structure. This section gives an informal overview of how the properties of a metric system can be used to organize data in efficient index structures.

2.5.1 Types of queries

Assume that we are given a query string of symbols, q, a database of stored strings, U, and a distance function $d(q,u)$ which for each $u \in U$ returns the distance between q and u. In [CNBYM99], Chávez et al. define three basic types of queries against U: *Range* queries, *nearest neighbour* queries and *k nearest neighbours* queries. Since the nearest neighbour query is the same as a k nearest neighbours query with $k = 1$, the basic types can be further generalized to the following two:

Range queries Find all elements in the database which are within a distance r to the query q. That is, find all $u \in U$ such that $d(q, u) \leq r$.

Nearest neighbours queries Find the k elements in the database which are closest to q. That is, find a set $R \subseteq U$ such that $|R| = k$ and $\forall v \forall w \{v \in R \wedge w \in (U - R) \rightarrow d(q, v) \leq d(q, w)\}$.

Several additional restrictions may be imposed, such as specifying a maximum distance from the query string in a nearest neighbours query.

It is clear that both range queries and nearest neigbour queries can be answered by comparing the query string with every string stored in the database. In general, the cost of evaluating the distance between two strings and the size of the database makes this approach unfeasible. The goal of indexing is to organize the strings of the database in such a way that it is not necessary to compare the query string to all of them. Such a structure, where a number of database strings can be discarded at each stage of the search process based on properties of the distance function, will be beneficial for both types of queries. Both of them seek to minimize the distance function over the database - the difference is merely that range queries keep the maximum distance fixed while nearest neighbours queries keep the size of the result set fixed.

2.5.2 Utilizing the triangle inequality

The idea to use the triangle inequality to exclude database elements which cannot satisfy a query was first introduced by Burkhard and Keller in [BK73]. Assume that we are interested in finding the database element which is closest to the query element q. At each step of the search algorithm, we keep track of the best match found so far, y. If a new element x is found at this step which satisfies $d(q, x) < d(q, y)$, we set $y \leftarrow x$. Prior to the search process, a *reference element* u is chosen. When searching, distance evaluations between database elements and u are used together with the triangle inequality in two *cutoff criteria*:

First cutoff criterion From $d(q, x) + d(x, u) \geq d(q, u)$ we see that $d(q, x) \geq d(q, u) - d(x, u)$. Thus, if $d(q, u) - d(x, u)$ is greater than the distance between q and the best match found so far, x can immediately be excluded.

Second cutoff criterion From $d(q, x) + d(q, u) \geq d(x, u)$ we see that $d(q, u) \geq d(x, u) - d(q, x)$. Thus, if $d(x, u) - d(q, u)$ is greater than the distance between q and the best match found so far, x can immediately be excluded.

These two cutoff criteria can be combined into a joint cutoff criterion:

Joint cutoff criterion Element x can be excluded from further consideration if $|d(x,u) - d(q,u)|$ is greater that the distance between q and the best match found so far.

The idea underlying many indexing structures is simply that, given a reference element u, distances between u and all other elements of the databases can be precomputed. These precomputed values can be stored in a structure which enables us to exclude elements from considerations without having to compute the distance to the query element. Algorithm 1, adapted from [BFM+96], gives a simple example of such a scheme.

Algorithm 1 Simple search algorithm

```
 1: procedure PREPROCESS($U$, $u$)                    ▷ Executed only once
 2:     $X \leftarrow \emptyset$
 3:     for all $x \in U$ do
 4:         $X \leftarrow X \cup d(u,x)$
 5:     $X \leftarrow$ SORT($X$)
 6: end procedure

 7: procedure QUERY($q$, $u$)
 8:     $V \leftarrow X$
 9:     $\epsilon \leftarrow d(u,q)$
10:     $t \leftarrow d(u,q)$
11:     $y \leftarrow u$
12:     $x \leftarrow$ BINARYSEARCH($V$, $t$)          ▷ Find $x$ with $d(x,u)$ closest to $t$
13:     while $|d(x,u) - d(q,u)| > \epsilon$ do
14:         if $d(q,x) < \epsilon$ then
15:             $y \leftarrow x$
16:             $\epsilon \leftarrow d(q,x)$
17:         $V \leftarrow V - x$
18:         $x \leftarrow$ BINARYSEARCH($V$, $t$)
19: end procedure                    ▷ Variable $y$ now holds element closest to $q$
```

In this algorithm, U denotes the initial database of elements. The procedure PREPROCESS is called once to a create a sorted list, X, of distances between the reference element u and every element in U. Thereafter, procedure QUERY can be used to search for the database element closest to query element q. As can be seen, variable x is initialized to the element which has $d(x,u)$ closest to $d(q,u)$. Because the list V is sorted, this can be found by a simple binary search. As long as the joint cutoff criterion does not hold, the algorithm proceeds by checking if x is the best match found so far. Then the element pointed to by x is subtracted from V, and x is updated to the database element with $d(x,u)$ next closest to $d(q,u)$. Actually, the latter step can be achieved by simply checking the neighbours of x in V, but a repeated application of the BINARYSEARCH function has been included in the algorithm for simplicity. Since $|d(x,u) - d(q,u)|$ can only increase, a point will eventually be reached where all the remaining elements of V can be discarded because the joint cutoff criterion holds.

Thus, algorithm 1 can be seen as a simple example of the general strategy to structure the data in a way which enables us retrieve the correct answer without having to evaluate the distances between the query element and all database elements.

2.5.3 Some illustrative index structures for range queries

Many methods for indexing metric spaces exist which are more efficient than the one presented in algorithm 1. This section includes some illustrative examples of common index structures. Readers interested in a more detailed introduction to searching in metric spaces should consult a more extensive survey like [CNBYM99].

BKT - Burkhard-Keller Tree

One of the first general methods for indexing data from metric spaces was presented by Burkhard and Keller in [BK73]. A Burkhard-Keller Tree (BKT) assumes a discrete distance function. Given a database U of elements, the first step of the indexing process is to select a reference element u. This first reference element represents the root of the BKT. Since the distance function is discrete, we have a finite set of distances, D, from u to all other elements of V. For each distance $d \in D$, we define $U_d = \{v \in U \,|\, d(v,u) = d\}$. Each set U_d represents a child node. The tree is completed by simply applying this method recursively on each U_d with more than one element.

Figure 2.9 shows the two first steps of the creation of a BKT. Element D is selected as the first reference element. The other elements are partitioned into four sets, where elements of each set are equally distant from D. Element E is then selected as the reference element for the set $\{E, G, H, I\}$, and the process is applied recursively. The corresponding first step of the BKT is shown in figure 2.9b, while figure 2.9c shows the second step.

Given a query element q and a range r, we check nodes to see if they can be excluded. When arriving at a node u in the tree, we know that this node and all its children nodes are at a distance $d(p,u)$ from u's parent node p. We then check if $|d(p,q) - d(p,u)| > r$. If this holds, u and the entire subtree below this node can immediately be excluded. To see this, note that $d(u,q) \geq d(p,q) - d(p,u)$ by the triangle inequality. Thus, if $|d(p,q) - d(p,u)| > r$, we know that $d(u,q) >$

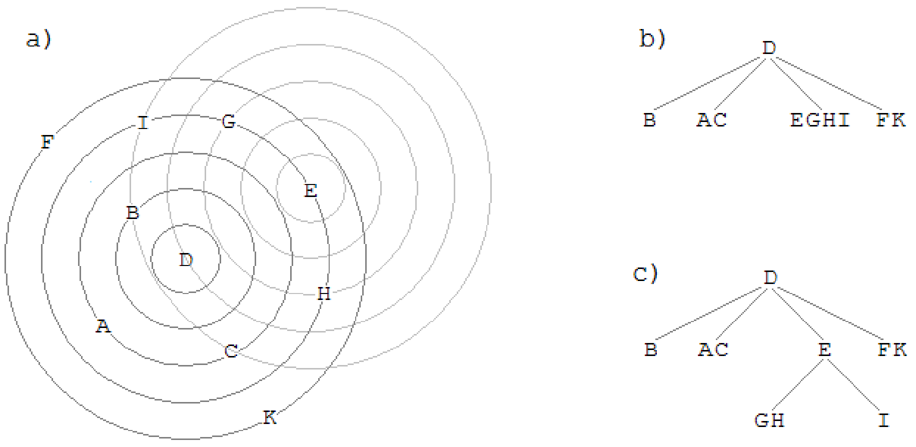


Figure 2.9: a) Two first steps of a BKT b) First step c) Second step

r. Because both u and all its children nodes are equally distant from p, this argument is used to exclude both u and its children from further consideration.

For elements which are not excluded, we calculate $d(q, u)$ and include element u in the result set if $d(q, u) \leq r$.

For a database of n elements, the construction complexity of BKTs is $O(n \log n)$. Query complexity is $O(n^{\alpha})$, where $\alpha < 1$ (i.e. sublinear). Numerous improvements to BKTs have been suggested in the literature. In [BYCMW94], a modification of BKT called FQT (Fixed Query Trees) is presented. The difference from BKTs is that the same reference element is used at each tree level, regardless of whether it belongs to a subset or not. The database elements are all stored at the leaves of the tree. The authors show by empirical experiments that this approach outperforms BKTs by performing less distance evaluations. Another variant of BKT is FHQT (Fixed Height FQT), presented in [BYCMW94] and [BY97]. In this data structure, all leaves are stored at the same depth. In addition, the same reference element is used at each tree level. It is shown that by making the tree deeper than necessary in this way, more distance evaluations can be avoided.

VPT - Vantage Point Tree

Data structures like BKT and FQT can only be used when the distance function is discrete, which is often not the case. Vantage-Point Trees (VPTs), presented in [Yia93] and [Chi94], are data structures applicable to continuous distance functions. They are built recursively in the following way. First, a reference element u is chosen as the root node. The distances from u to all other elements of the database U is then calculated, and the median distance m is found. Then the remaining elements of U are divided into two sets $L = \{p \in U \,|\, d(p, u) \leq m\}$ and $R = \{p \in U \,|\, d(p, u) > m\}$, and the procedure is applied recursively on those sets.

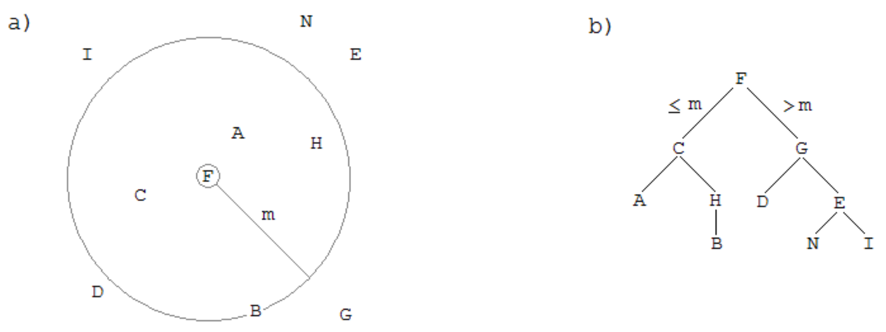


Figure 2.10: a) First step of the construction of a VPT b) Final tree

Figure 2.10a shows an example of the first step of the creation of a VPT. Element F is chosen as the root node. The median distance m ($d(F, B)$ in this case) is found, and the remaining elements are partitioned into two sets. The final VPT is shown in figure 2.10b.

Given a query q and a range r, the result set is found in the following way. Starting at the root node p, we first check if $d(q, p) < r$. If so, p is added to the result set. We then check if $d(q, p) - r \leq m$. If this is the case, the procedure is applied recursively on the left subtree of p. If not, the entire left subtree can be discarded from further consideration. Similarly, if $d(q, p) - r > m$, we enter the right subtree. If not, the right subtree can be discarded. Note that we may have to check both the left and the right subtree.

For very small ranges r, [Yia93] argues that query complexity on VPTs is $O(\log n)$. However, as r increases, the number of times we have to enter both the left and the right subtree also increases. So the argument that VPT query processing is $O(\log n)$ is actually only true when searching for exact matches ($r = 0$).

As with BKT, numerous improvements have been suggested for VPT. In [BO97], the MVPT (Multi Vantage Point Tree) is introduced. A MVPT is an m-ary tree, where $m > 2$ (in contrast to the binary VPT). Empirical experiments show that MVPTs outperform VPTs in some cases. Other advantageous modifications of VPT can be found in [CNBYM99].

2.5.4 Index structures for nearest neighbour queries

The index structures presented in section 2.5.3 are designed for range queries. There exist several techniques for using these structures to answer nearest neighbour queries. The simplest possible scheme is to start with range $r = 1$ and increase r until the size of the result set is appropriate. A common way of increasing r is to start with $r = a^i$ and, if the size of the result set is too small, set $r = a^{i+1}$. For the first iteration we set $i = 0$. When a point is reached where the size of the result set is too big, the range is refined between a^{i-1} and a^i.

The problem with such schemes is that the complexity of the range queries grows as the range grows. Thus, several methods have been suggested for limiting both the number of range queries and the size of the largest range used in the search. Again, readers are referred to [CNBYM99] for an in-depth survey.

2.5.5 The challenge of non-metricity

The index structures presented in the previous section rely on the triangle inequality for excluding database elements from consideration without having to compute the distance to the query element. If the distance function is not metric, queries on such structures will not produce correct results.

Some simple techniques have been suggested for dealing with distance functions where the triangle inequality does not hold. In [CNBYM99], it is suggested that if one can find constants α, β and δ such that the triangle inequality can be relaxed to $d(x,y) \leq \alpha d(x,z) + \beta d(z,y) + \delta$, the algorithms designed for metric spaces can be modified to handle this new inequality. There are two problems with this approach. Firstly, we may not be able to specify such constants at a general level for the distance function at hand. Secondly, the number of exclusions we can make during query processing decreases rapidly as the constants increase. To see this, assume that we are given an α such that $d(x,y) \leq \alpha[d(x,z) + d(z,y)]$ holds, and that the data is indexed in a BKT. Given a query element q and a node element u, recall from section 2.5.3 that we calculate $|d(p,q) - d(p,u)|$ (where p is u's parent) and eliminate the entire subtree under p if this value is greater than the range r. However, the existence of the constant α makes it necessary to check $|\frac{d(p,q)}{\alpha} - d(p,u)| > r$, which holds in a decreasing number of cases as α increases. Thus, query processing performance degrades sharply when the triangle inequality is relaxed in this way.

Therefore, much research effort has recently focused on finding efficient mappings from non-metric to metric distance functions. A number of requirements could be stated for such a mapping:

- The distances produced by the mapping should be as close as possible to the original distances.
- The number of misclassification made by the distance function produced by the mapping should be as small as possible. In other words, if the original distance function classifies element B as being closer to element A than element C is, this should ideally also be the case for all triples of elements (A, B, C) with the new distance function.
- The distance function produced by the mapping should underestimate the original distance function. If this is the case, we are guaranteed the correct answer to a range query. The index is first used to obtain an intermediate

answer. Then the solution set is refined using the original distance function.

Finding such a mapping means that efficient methods like the ones presented in the previous sections can be used to index the data, thereby providing sublinear access to the data. This is very desirable in cases where large amounts of data are available, such as protein databases. However, the problem has proved to be very challenging since there exists no direct mapping from the similarity scores of mutation matrices like PAM and BLOSUM to a metric distance function which preserves all the information contained in the matrix. The methods proposed, presented and compared in this work aims at producing metric distance matrices where the information loss is minimal.

2.6 Related work

The problem of deriving a metric distance matrix from existing mutation matrices has been previously been adressed in a few different ways.

In [XM04], the matematics of the original PAM series of mutation matrices have been reworked. Instead of using the empirically derived mutation frequency data to derive a matrix which gives the probability of amino acid i changing to amino acid j within a number of evolutionary steps, this work attempts to derive a metric distance between amino acids by considering the expected *time period* between a transition from amino acid i to amino acid j. Using such a distance measure, amino acid pairs with a high mutation probability will be associated with a short time interval, thus providing a dissimilarity function instead of a similarity function. The method prodces a distance matrix with minimal metric disortions. Some small manual adjustments results in the mPAM250 matrix, which is shown empirically to retain a large fraction of the sensitivity of PAM250 when used in a homology search. There are two main problems with this approach. Firstly, it is specific to the PAM series of matrices, since it is based on the mathematical foundation of this model. Secondly, the derived distance matrices are not guaranteed to be metric, requiring manual adjustments for the triangle inequality to be satisfied. In essence, this brings us back to the original problem.

Halpering et al. introduce a metric distance mapping for the BLOSUM family of matrices in [HBK$^+$03]. In their work, peptides (short sequences of amino acids) are mapped into bit vectors. The distance between any two peptides can then be approximated by the Hamming distance between the bit vectors, which is a metric. The problem is slightly different from the problem addressed in this work, because the method is applied to protein data directly rather than being used to produce a metric distance matrix. Furthermore, the method is specific to the BLOSUM series of matrices.

Dhillon et. al state in [DST04] that their *triangle fixing algorithm* was originally motivated by the need to convert amino acid substitution matrices into metric distance matrices. This algorithm is described in detail in chapter 3. It has been implemented in this work as a point of comparison for the other methods and algorithms proposed. The problem with the algorithm in the form it is presented in the original paper is that it accepts a non-metric *distance* metric as input. Thus, we are faced with the question of how to convert a similarity matrix into a distance matrix.

Taylor and Jones have previously proposed and compared a number of methods for projecting substitution matrices into low-dimensional metric spaces in [TJ93]. They also propose an *inter-row* method of transforming the PAM250 matrix into a metric distance matrix, which performs relatively well when being used for homology search. The work by Taylor and Jones is one of the main inspirations behind methods proposed and evaluated in this work.

Chapter 3

Methods and solutions

As shown in chapter 2, the properties of metric distance functions are very desirable, since they enable indexing of the data at hand. Many distance functions do not possess these properties, so algorithms for converting non-metric substitution matrices into metric ones are also very desirable. Assuming that the distance functions are given as matrices, the metric distance matrix produced by such an algorithm should ideally preserve all the sensitivity of the original non-metric one. A sensitive distance matrix will produce result sets with large fractions of true positive hits and small fractions of false positive hits when used for searching in a database of elements. This raises a number of questions:

- What algorithms can be used to convert the non-metric matrix into a metric distance matrix?
- Given a matrix with similarity values instead of distance values, how can this fact be incorporated into the algorithms?
- Which algorithm produces the most sensitive metric distance matrix?
- Which factors determine the sensitivity of a metric matrix relative to the original non-metric one? How are these factors controlled in the algorithms?
- How efficient are the algorithms in terms of computational resource usage? Can they be used to convert large non-metric matrices into metric ones?

To answer these questions, a number of methods and algorithms for generating metric distance matrices from non-metric substitution matrices are presented in this chapter. Although tested and compared in the field of bioinformatics, the methods are intended to be generic to the problem of generating metric distance matrices from non-metric ones. Two main approaches to solving the problem are presented. Section 3.1 presents an existing algorithm for converting a non-metric distance matrix into a metric one directly. The main idea underlying this work an approach where N elements (e.g. amino acids) are mapped into a metric space, from which a metric distance matrix is thereafter recovered. Section 3.2 presents this idea in general, while sections 3.3 to 3.6 are devoted to specific algorithms. Some of the suggested solutions are based on existing algorithms, while others are unique to this work:

- Multidimensional scaling (section 3.3), FastMap (section 3.4) and BoostMap (section 3.5) are existing algorithms which have been adopted in this work for solving the problem of mapping N elements into a metric space.
- Some modifications specific to the problem at hand are proposed for multidimensional scaling in section 3.3.2 and for BoostMap in section 3.5.3.
- Based on potential limitations of the other methods, two genetic algorithms for solving the problem are proposed in section 3.6.

Finally, section 3.7 rounds off the chapter with a summary of the proposed methods and and how they solve the problem. The main conclusions are drawn in chapter 4 and 5, where experimental results are presented and discussed, respectively. Also note that algorithms will be analyzed and compared in terms of computational complexity in chapter 5 rather than in this chapter.

3.1 The triangle-fixing algorithm

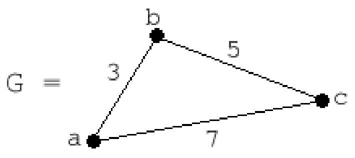
The *metric nearness problem* has been defined by Dhillon et al. in [DST04]. Given a non-metric matrix D of dissimilarity measures, the problem is to find the "nearest" matrix of dissimilarity values which satisfies the triangle inequality. To solve this problem, a class of *triangle fixing* algorithms has been developed for the L_p norms (see section 2.4.2). An explicit algorithm is given for the L_2 norm. It works by iteratively enforcing the triangle inequality for each violated triangle. The pseudocode is shown in algorithm 2, which has been adopted directly from [DST04].

In this algorithm, the dissimilarity measures are seen as edges of a graph G. Associated with the graph is a matrix A representing all possible triangles. T is simply a list of all triangles, corresponding to the rows of A. An example is shown in figure 3.1. In each row of T, the number 1 indicates that the edge is the hypotenuse of the triangle. If the triangle inequality is not satisfied for a specific triangle, the corresponding element of Td' will be negative.

In each iteration of the algorithm, all triangles are looped through and corrected. In essence, Dykstra's algorithm (presented in [Dyk83]) is used to minimize $\frac{1}{2}||x^2||$ subject to $Ax \leq -Ad$ in lines 7 to 16 of the algorithm. In line 7, ab represents the column of A labeled 1, while bc and ca represent the columns labeled -1. The mathematical derivations necessary to arrive at steps 7 to 16

Algorithm 2 TRIANGLEFIX

```
Input: $G, T, \epsilon$
 1: for each $(i,j,k) \in T$ do
 2:     $\delta_{ijk} = 0$                                   ▷ Initialize correction terms
 3: $\Delta \leftarrow \epsilon + 1$
 4: while $\Delta > \epsilon$ do
 5:     $\Delta \leftarrow 0$
 6:     for each $(a,b,c) \in T$ do
 7:         $(oa, ob, oc) \leftarrow (ab, bc, ca)$
 8:         $ab \leftarrow ab + \delta_{abc}$                 ▷ Apply correction
 9:         $bc \leftarrow bc - \delta_{abc}$
10:         $ca \leftarrow ca - \delta_{abc}$
11:         $\delta \leftarrow ab - bc - ca$
12:         if $\delta > 0$ then
13:             $ab \leftarrow ab - \delta/3$
14:             $bc \leftarrow bc + \delta/3$
15:             $ca \leftarrow ca + \delta/3$
16:         $\delta_{abc} \leftarrow \delta_{abc} - ab + oa$
17:         $\Delta \leftarrow |oa - da|$
18: return corrected $G$
```



$$d = [3 \quad 7 \quad 5]$$

$$A = \begin{array}{c|ccc} & ab & ac & bc \\ \hline abc & 1 & -1 & -1 \\ bca & -1 & -1 & 1 \\ cab & -1 & 1 & -1 \end{array}$$

Figure 3.1: Notation used in the triangle fixing algorithm

of the algorithm are considered to be outside the scope of this work. Here it is sufficient to know that the steps add a correction term to each edge of G. The correction terms together correspond to one iteration of Dykstra's solution to the minimization problem. Readers are referred to [DST04] for details on this derivation.

The algorithm terminates and returns the corrected graph G when no triangle receives a significant update. A corrected distance matrix can then be found simply as the edges of the corrected graph G. If ϵ is chosen low enough to reach some degree of convergence, the corrected distance matrix will be metric.

In this work, the L_2 triangle fixing algorithm has been implemented to test its ability to derive a metric distance matrix from a non-metric substitution matrix. The fact that substitution matrices like PAM and BLOSUM (see section 2.3) are *similarity* matrices poses a problem, since the triangle fixing algorithm takes a *dissimilarity* matrix as input. Amino acid substitution matrices must therefore be transformed into distance matrices before the algorithm can be applied. The transformation specified in equation 2.12 has been used in this work.

The fact that the substitution matrix must be transformed into a distance matrix is a potential problem. There is no straightforward way of converting a similarity matrix, like an amino acid substitution matrix, into a distance matrix with zeros on the diagonal without losing information. Section 2.3.3 elaborates further on this. The algorithm finds the best metric fit to a distance matrix, but it is questionable how good this matrix reflects the data in the original similarity matrix in the first place. This suggests that finding methods for deriving a metric distance matrix using the data in the substitution matrix *directly* could prove to be advantageous.

Nevertheless, the triangle fixing algorithm is included as a possible solution to the problem of making substitution matrices metric. The next section will present a conceptually different approach, where metric matrices are derived by first mapping elements (for example amino acids) into a metric space. Analytical and empirical comparisons of the two approaches will be given in chapter 5.

3.2 Deriving metrics from spatial embeddings

Recall from section 2.4 that a *space* can be defined as a pair (S, d) of a set S of elements and a distance function $d : S \times S \rightarrow \mathbb{R}$. The distance function determines if the space is metric or not. Based on this definition, the problem of deriving a distance metric can be approached using the general idea to construct a new distance matrix from N points embedded in a metric space. Three basic steps are necessary for this scheme:

1. Define a metric space.
2. Given an $N \times N$ input matrix (possibly non-metric), find the embedding of N elements in this space which minimizes some penalty function.
3. Use the distance function of the metric space to generate a metric output matrix.

In the following sections, a number of methods are presented which use this scheme of deriving distance metrics using spatial embeddings. Section 3.3 presents a solution based on *multi-dimensional scaling*. Section 3.4 shows how the *FastMap* algorithm can be used. In section 3.5 it is shown how the *BoostMap* algorithm can be used in the case when the input matrix consists of dissimilarity values, while section 3.5.3 presents some advantageous modifications which can be done to the algorithm when the input matrix is a similarity matrix (e.g. an amino acid substitution matrix). Finally, section 3.6 proposes two genetic algorithms for generating a metric distance matrix.

3.3 Multi-dimensional scaling

3.3.1 Metric and non-metric multidimensional scaling

Multidimensional scaling (MDS) techniques have been known for decades, and are among the most commonly used methods for finding an embedding of objects corresponding to the dissimilarity relations between them. The basic idea is to find a constellation of points in d dimensions corresponding as close as possible to the data at hand. The term "scaling" stems from the fact that such a transformation can be seen as a scaling of points in a space with unknown dimensionality. Multidimensional scaling techniques are sometimes called *dimensionality reduction* techniques, because the unknown space in which the data is embedded is assumed to have higher dimensionality than the d-dimensional space which it is to be embedded into. There a two basic types of multidimensional scaling:

Metric MDS makes the assumption that dissimilarity measures between objects to be embedded are available. It is called "metric MDS" because the transformation assumes that the metricity of the data should be preserved, i.e. it implicitly assumes metricity of the data at hand.

Non-metric MDS makes the assumption that the rank order of pairs of objects are available; i.e. it is known for all (i, j) and (k, l) whether $D_{i,j} < D_{k,l}$ or not. A scaling based on rank orders does not necessarily preserve the assumed metricity of the original space, hence the name "non-metric MDS". In fact, the method makes no assumption as to whether the data at hand is metric or not.

As briefly described in section 2.3.3, converting a similarity matrix into a dissimilarity matrix, metric or not, without loss of information is problematic. Non-metric MDS therefore seems more appealing than metric MDS, since the latter technique assumes that dissimilarities between objects are available. Furthermore, the metricity preserving transformation of metric MDS may be problematic if the dissimilarity measures at hand are not metric. This is true in most cases when equation 2.12 is used to transform an amino acid similarity matrix into a dissimilarity matrix.

Thus, non-metric MDS (hereafter referred to as NMDS) has been considered as a candidate solution to the problem of recovering a metric from a metric space. An important observation to make is that rank order of pairs of elements can be derived directly from a similarity substitution matrix rather than from an intermediate dissimilarity matrix; sorting dissimilarity values ascendingly is the same as sorting similarity values descendingly. Thus, it should not be necessary to use equation 2.12 to convert the substitution matrix into a dissimilarity matrix first. This independence from an intermediate dissimilarity matrix makes NMDS especially appealing.

Most NMDS algorithms work in iterations, where a monotonic transformation of the current point constellation is applied to ensure that distances between points are monotone with respect to the original ranking. Further details on this can be found in [GCS70]. For technical reasons, this monotonic transformation has traditionally been implemented by using the data in the distance matrix to generate a set of "pseudo-distances". As noted by Taguchi and Oono in [TO04], this dependence on the data of the distance matrix conflicts with the idea that only the rankings of distance (or similarity) measures should be required. They propose a "pure" NMDS algorithms which does not depend on data in any distance matrix. This independence from intermediate distance measures, together with the idea to rank object pairs according to similarity value, suggests that this algorithm should be the ideal multidimensional scaling technique for the problem at hand. It has previously been applied in the field of bioinformatics to analyse genes from microarray data [TO05].

3.3.2 NMDS with proposed modification

Algorithm 3 is a slightly modified version of the "pure" NMDS algorithm presented in [TO04]. The modification is that an $N \times N$ similarity matrix S is used to produce the ranks of pairs of objects instead of a distance matrix. In line 1, a list G of ranked pairs (i, j) of objects is generated. The pair with highest similarity value is stored at index 0, the pair with the next highest similarity value is stored at index 1, and so on. The N objects (e.g. amino acids) to be embedded are represented as d-dimensional position vectors r_i, $i \in \{1, ..., N\}$. At each iteration of the algorithm, the L_1 (Manhattan) distances are computed between position vectors. These distances are then ranked ascendingly, and differences in positions in G and H are calculated for all pairs (i, j). The values $C_{i,j}$ are measures of how closely ranked a pair is in G and H. They are used in line 13 to calculate displacement vectors for all N position vectors. It it shown in [TO04] that these displacement vectors are directed towards constellations of less potential energy. The parameter s is an empirically derived scaling constant, typically set to $s = N^{-3}$.

The algorithm iterates until some convergence goal is reached. Such a goal may be that the sum of magnitudes of the displacement vectors (the "potential energy") is sufficiently small, that the mean of $C_{i,j}$ is sufficiently small, or that a maximum number of iterations has been reached.

As noted in the original paper, the algorithm is not free from the problem of local minima. However, when dealing with a relatively small number of points, it is affordable to run the algorithm repeatedly. The best embedding is then chosen as the one where the $C_{i,j}$ shows the least difference in ranks. For the purpose of mapping the data of 20×20 amino acid substitution matrices into d dimensions to obtain a metric distance matrix, the SIMILARITYNMDS algorithm has been implemented in Matlab. The source code is included on the CD which should

Algorithm 3 SimilarityNMDS

```
Input: $S, d, s$
 1: $G \leftarrow [... \geq S_{i,j} \geq S_{k,l} \geq ...]$
 2: Generate $N$ random position vectors $r_i$ in $\mathbb{R}^d$
 3: while not converged do
 4:     Scale each position vector $r_i$ in $\mathbb{R}^d$ so that $\sqrt{\sum_i |r_i|^2} = N$
 5:     for all pairs $(i, j)$ do
 6:         $\delta_{i,j} \leftarrow |r_i - r_j|$
 7:     $H \leftarrow [... \leq \delta_{i,j} \leq \delta_{k,l} \leq ...]$
 8:     for all pairs $(i, j)$ do
 9:         $n \leftarrow$ index of $D_{i,j}$ in $G$
10:         $m \leftarrow$ index of $\delta_{i,j}$ in $H$
11:         $C_{i,j} \leftarrow n - m$
12:     for $i \in \{1, ..., N\}$ do
13:         $r_i \leftarrow r_i + s \sum_j C_{i,j}(r_i - r_j) \ / \ |r_i - r_j|$
14: for all pairs $(i, j)$ do
15:     $D_{i,j} \leftarrow |r_i - r_j|$
16: return $D$
```

accompany this report.

3.4 FastMap

FastMap is an algorithm for mapping points into a d-dimensional space which has gained much popularity since its introduction by Faloutsos et al. in [FL95]. Like MDS, FastMap is called a *dimensionality reduction* technique, because it solves the problem of mapping points from a space of unknown dimensionality into a d-dimensional space while preserving the dissimilarities as good as possible. It solves the same problem as MDS, but with an order of magnitude lower computational complexity. Like metric MDS, FastMap assumes that N^2 dissimilarities between N objects are specified, and that these dissimilarities satisfy the triangle inequality. As will be explained below, the algorithm works by recursively projecting points into spaces of lower dimensionality. The triangle inequality must be satisfied for these projections to be correct. However, the resulting distance matrix derived from the embedding is assured to be metric if a metric distance function is chosen for the d-dimensional space. Therefore, FastMap is included as a candidate solution to the problem of deriving a metric distance matrix from a mapping of N points in a d-dimensional space. The effects of providing non-metric data to the algorithm are not known in advance; experimental results will uncover their impact on the ability of the algorithm to preserve the information in the original matrix. FastMap has previously been applied to non-metric data

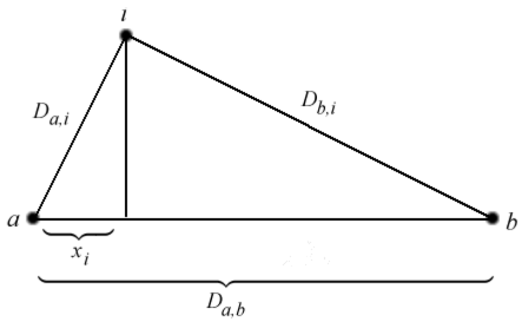


Figure 3.2: Illustration of the cosine law

in [YJF98].

The main idea behind FastMap is to to recursively map points from a space of dimensionality d into a hyperplane in this space of dimensionality $d-1$. As an example, assume that we are given a set of points in two dimensions. Two points, a and b, are picked as *pivot* points, and a line is drawn between them. This line can be seen as a one-dimensional hyperplane in the two-dimensional space. All remaining points are projected onto this line using the *cosine law*. For a point i, the distance to the first pivot point can be found using equation 3.1, where D is a matrix containing the distances between points in the two-dimensional space. The equation is illustrated in figure 3.2.

$$x_i = \frac{D_{a,i}^2 + D_{a,b}^2 - D_{b,i}^2}{2D_{a,b}} \tag{3.1}$$

As shown in [FL95], this equation can be used in the general case for mapping points from any d-dimensional space into a $d-1$-dimensional hyperplane. The same paper also given a formal proof of equation 3.2, which specifies how a new distance function can be computed for the projected points in the $d-1$-dimensional hyperplane.

$$D'_{i,j} = \sqrt{D_{i,j}^2 - (x_i - x_j)^2} \tag{3.2}$$

The idea of FastMap is as follows. Given an $N \times N$ distance matrix D, we pretend that the N points are indeed points in d dimensions, and use equation 3.1 to project the points into a $d-1$-dimensional hyperplane. For each point i, the x_i value is stored as the first coordinate of the final d-dimensional embedding. A new distance matrix is then computed for the projected points using equation 3.2, and the entire procedure is repeated to obtain the second coordinate of the final embedding. We continue recursively in this way until a one-dimensional projection is reached. This projection then represents the last coordinates of the d-dimensional embedding.

$$D = \begin{bmatrix} 0 & 1 & 1 & 100 & 100 \\ 1 & 0 & 1 & 100 & 100 \\ 100 & 100 & 100 & 0 & 1 \\ 100 & 100 & 100 & 1 & 0 \end{bmatrix} \qquad X = \begin{bmatrix} 0 & 0.707 & 0.668 \\ 0.005 & 1.414 & 0.935 \\ 0.005 & 1.061 & 0 \\ 100 & 0.707 & 0.668 \\ 99.995 & 0 & 1 \end{bmatrix}$$

Figure 3.3: Initial distance matrix (D) and final embedding (X)

An example is shown in figure 3.3, where D is the initial distance matrix and X is the final embedding produced by FastMap. The example has been adopted from [FL95]. Clearly, $d = 3$ has been selected in this case. The rows of X represent the the N embeddings, and each column of a row represents a coordinate. Some readers may observe a similarity to principal component analysis (PCA), where the principal components are decreasingly capable of preserving information. The same can be said about the embedding produced by FastMap, where the projection into the first hyperplane (of dimensionality $d-1$) preserves the largest fraction of information.

Algorithm 4 FASTMAP

```
Input: $d, D$
Extern: $X, c$                                        ▷ $X$ and $c$ are global variables
 1: if $d = 0$ then
 2:     return $X$
 3: $c \leftarrow c + 1$
 4: $a, b \leftarrow$ CHOOSEDISTANTOBJECTS($D$)
 5: if $D_{a,b} = 0$ then
 6:     for $i \leftarrow 1, N$ do
 7:         $X_{i,c} \leftarrow 0$
 8:     return $X$
 9: for $i \leftarrow 1, N$ do                         ▷ Project points using the cosine law
10:     $X_{i,c} \leftarrow (D^2_{a,i} + D^2_{a,b} - D^2_{b,i}) \,/\, 2D_{a,b}$
11: for $i \leftarrow 1, N-1$ do                       ▷ Generate new distance function
12:     for $j \leftarrow i+1, N$ do
13:         $D'_{i,j} \leftarrow \sqrt{D^2_{i,j} - (X_{i,c} - X_{j,c})^2}$
14:         $D'_{j,i} \leftarrow D'_{i,j}$
15: return FASTMAP($d-1, D'$)
```

Pseudocode for FastMap is shown in algorithm 4. As explained, the algorithm calls itself recursively. Variables X and c are global to all recursive calls. X stores the coordinates of the final embedding, while c simply points to the appropriate column of X. Line 1 checks if the algorithm has reached a point of zero dimensionality, which means that the final embedding can be returned. In line 4, two pivot points are chosen for the current recursive call using the external procedure ChooseDistantObjects. This procedure is explained in detail below. Lines 5 to 8 simply check if the distance between the two pivot points is zero, which means that all distances between all points are zero (because the two pivot points should ideally be the two points farthest apart in the current space). If this is the case, there is no point in continuing, so the coordinate values are set to zero and the resulting embedding is returned. Lines 9 to 10 project all points onto a hyperplane of dimensionality $d-1$, and lines 11 to 14 calculates a new distance matrix for the projected points. Finally, the recursive call is made in line 15 with a decremented dimensionality counter and the new distance matrix as arguments.

It should be noted that the calculation of a new distance matrix in lines 11 to 14 is not strictly necessary. Instead, distance values could be computed on the fly for a specific d using equation 3.2 and X in lines 9 to 10. Thus, the pseudocode is not the most efficient one, but the most illustrative one in terms of understanding the algorithm.

Using the final embedding returned by the algorithm, any metric distance function can be used to generate a metric distance matrix. For example, the euclidean distance between two points i and j can be calculated as follows:

$$\delta_{i,j} = \sqrt{(X_i - X_j)(X_i - X_j)^{\mathrm{T}}} \tag{3.3}$$

where X_i and X_j are row vectors of X.

Algorithm 5 ChooseDistantObjects

```
Input: $D$
1: $b \leftarrow$ randomly selected number from $\{1, ..., N\}$
2: $a \leftarrow \text{argmax}_{a \in \{1,...,N\}} D_{a,b}$
3: $b \leftarrow \text{argmax}_{b \in \{1,...,N\}} D_{a,b}$
4: return $a, b$
```

A question which remains to be answered is how to pick the pivot points a and b. All other points are to be projected onto a hyperplane perpendicular to the line between these two points, so it is intuitively desirable to choose the two points farthest apart. The straightforward way of finding the two points farthest apart clearly has a complexity which is quadratic in N. This is not ideal in cases where N is large, so a heuristic algorithm which is linear in N is proposed in the original paper. Pseudocode is shown in algorithm 5. The algorithm starts by initializing b to a random point. Then a is selected to be the point farthest apart from b. This can clearly be achieved in $\Theta(N)$ time. Finally, b is updated to the point farthest part from a. Points a and b should now intuitively be a good candidate solution to the problem of finding the two points separated by the greatest distance. As noted in the original paper, the two middle steps of the algorithm can be repeated a constant number of times while still preserving a complexity linear in N.

In this work, FastMap is compared with other algorithms for generating a metric distance function from a metric embedding. Since FastMap requires a matrix of distances to be specified, a similarity matrix like an amino acid substitution matrix must fist be converted into a dissimilarity matrix before the algorithm can be applied. Thus, the necessary steps to generate a metric distance matrix from a substitution matrix are:

1. Convert similarity matrix S into a dissimilarity matrix D using equation 2.12.
2. $X \leftarrow \text{FastMap}(d, D)$.
3. Generate a metric distance matrix O from X using a metric distance function. The L_2 norm was found to produce the best matrices in this work.
4. Return O.

Two points should be made regarding the steps presented above:

- The preservation of information is questionable when converting a similarity matrix into a dissimilarity matrix using a transformation such as equation 2.12. This is explained in detail in section 3.5.3, where the concept of triangular misclassifications is introduced.
- The result of converting a similarity matrix like an amino acid substitution matrix into a dissimilarity matrix using equation 2.12 is not metric. As explained above, FastMap assumes data which satisfy the triangle inequality, and it is uncertain how it will perform when provided with a non-metric distance matrix.

Nevertheless, FastMap represents a straightforward approach to the problem of mapping N points into a d-dimensional space using an $N \times N$ distance matrix. Metric multidimensional is a classical approach to this problem, but FastMap was chosen because it essentially solves the same problem with an order of magnitude lower computational complexity. The Matlab source code of the implementation is included on the enclosed CD.

3.5 BoostMap

BoostMap is an algorithm introduced recently by Athitsos et al. in [AASK04b] and [AASK04a]. It is used to map both metric and non-metric data into a metric space. This makes it highly interesting as a possible solution to the problem of mapping non-metric substitution matrices into metric space. BoostMap is based on the more general optimization algorithm AdaBoost, which is presented first.

3.5.1 AdaBoost - Adaptive Boosting

BoostMap builds upon AdaBoost (Adaptive Boosting), an algorithm introduced in [FS97] for improving the accuracy of prediction rules. In general, *boosting* refers to the strategy of combining many less accurate prediction rules into a highly accurate combined prediction rule. In AdaBoost, such a prediction rule is a *classifier* h_j. Assume that we are given a set $\{x_1, ..., x_t\}$ of entities x_i of some kind. The entities may be scalars, vectors or any kind of data structure. Given such an entity x_i as input, the classifier h_j outputs a value from the set $\{-1, 1\}$. For each entity x_i, we are also given a label $y_i \in \{-1, 1\}$. This label makes is possible to measure the accuracy of the classifier h_j; a perfect classifier would output $h_j(x_i) = y_i$ for all x_i.

A core idea of the AdaBoost algorithm is to maintain a distribution w of weights, where $w_{i,j}$ corresponds to the weight of the pair (x_i, y_i) in training round j. The set of pairs (x_i, y_i) is called a *training set*. At each iteration of the algorithm, a *weak learning algorithm* is called. Such a weak learning algorithm takes the current weights of the training set as input, and outputs a weak classifier h_j for training round j. The accuracy of this classifier is then calculated, and a parameter α_j is assigned to it based on the accuracy. Then the training weights are updated. Weights of incorrectly classified entities are increased, so that the weak learning algorithm is forced to focus on these the next training round. Intuitively, this would cause the weak learning algorithm to choose a weak classifier which corrects the "mistakes" done by other weak classifiers.

A total of J weak classifiers are selected. The final strong classifier is found by taking the sign of the sum of output values from the weak classifiers, where each classifier is weighted by its importance parameter α_j. Pseudocode for AdaBoost is shown in algorithm 6.

Algorithm 6 ADABOOST, high-level pseudocode

```
Input: $(x_1, y_1), ..., (x_t, y_t)$                          ▷ $x_i \in X, y_i \in \{-1, 1\}$
Extern: WEAKLEARNER
1: for $i \leftarrow 1, t$ do
2:     $w_{i,1} = \frac{1}{t}$
3: for $j \leftarrow 1, J$ do
4:     Call WEAKLEARNER($\{w_{i,j} \mid i \in \{1, ..., t\}\}$)
5:     Get weak classifier $h_j$
6:     Calculate the error of $h_j$
7:     Choose an optimal $\alpha_j \in \mathbb{R}$
8:     Set $w_{i,j+1} = w_{i,j} \exp(-\alpha_j y_i h_j(x_i))/z_j$        ▷ $z_j$ is a normalization factor
9: return $H(x) = \text{sign}(\sum_{j=1}^{J} \alpha_j h_j(x))$
```

The full details of significance of the weight update factor (line 8) can be found in [SS99].

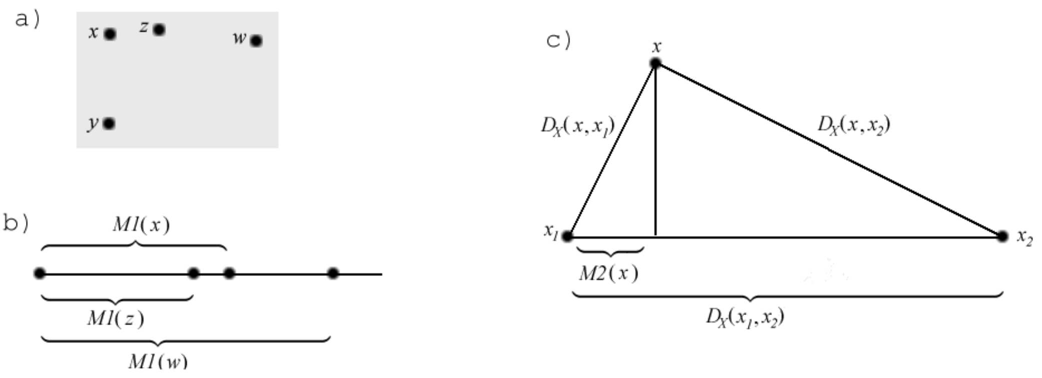


Figure 3.4: a) Two-dimensional points b) M1 embedding of two-dimensional points c) Calculation of M2 embedding of a point

3.5.2 BoostMap - using AdaBoost to produce embeddings

The main idea of BoostMap is to use one-dimensional embeddings as classifiers, and combine many such one-dimensional embeddings into a high-dimensional embedding with a low error rate in relation to the given distance matrix. In essence, this is achieved as follows.

- Given an $N \times N$ distance matrix, the goal is to find a metric embedding of N objects $Z = \{z_1, ..., z_N\}$.
- The set X consists of triples $x_i = (z_a, z_b, z_c)$ of distinct objects.
- A proximity classifier $P_X(z_a, z_b, z_c)$ is defined. P_X outputs 1 if z_c is closer to z_a than z_b is, -1 if z_b is closer to z_a than z_c is or 0 if z_b and z_c are equally close to z_a. For a given triple $x_i = (z_a, z_b, z_c)$, the associated label is $y_i = P_X(z_a, z_b, z_c)$.
- Each one-dimensional embedding $M(x)$ has a proximity classifier $h(x) = h(z_a, z_b, z_c)$ with the same interpretation as P_X.
- Using the AdaBoost algorithm presented in the previous section, weak classifiers from one-dimensional embedding are chosen by a weak learner. The corresponding one-dimensional embeddings are then combined into a high-dimensional embedding. Weak classifiers are weighted so that the number of misclassifications done by the high-dimensional embedding in relation to the P_X values is minimized.

Two types of one-dimensional embeddings of objects are specified: $M1$ and $M2$. Given an $N \times N$ distance matrix D, both $M1$ and $M2$ can be calculated in $\Theta(N)$ time. For an $M1$ embedding, a single reference object r is selected. The embeddings of all other objects are then calculated as $M1_r(x) = D_{x,r}$. For example, consider the set of two-dimensional objects shown in figure 3.4a. If y is selected as reference object, we obtain the embedding $M1_y$, which is shown in figure 3.4b. Figure 3.4 has been reproduced and slightly modified from [AASK04a].

The idea behind $M2$ is to draw a line between two reference objects x_1 and x_2. One-dimensional embeddings of all other objects are then found as the orthogonal projections of the objects onto this line. They are calculated as shown in equation 3.4, which is illustrated in figure 3.4c. Observe that this is exactly the same one-dimensional embedding as used by FastMap.

Each $M1$ or $M2$ embedding has a classifier $h(x)$ associated with it. The classification error of $h(x)$ is expected to be high, but nevertheless better than a random classifier since the embeddings are based on data from the distance matrix.

$$M2_{x_1,x_2} = \frac{D^2_{x,x_1} + D_{x_1,x_2} - D^2_{x,x_2}}{2D_{x_1,x_2}} \tag{3.4}$$

To evaluate the accuracy of a classifier, Schapire et al. has defined the function Z_j which gives a measure of the appropriateness at iteration j of the algorithm. The function is shown in equation 3.5.

$$Z_j(h,\alpha) = \sum_{i=1}^{|T|} w_{i,j} \exp(-\alpha y_i h(z_{a,i}, z_{b,i}, z_{c,i})) \tag{3.5}$$

In this function, T is the training set $\{x_1, ..., x_{|T|}\}$, where a triple x_i is represented as $(z_{a,i}, z_{b,i}, z_{c,i})$. If y_i and $h(x_i) = h(z_{a,i}, z_{b,i}, z_{c,i})$ has the same value, the product between them equals one, thereby contributing less to Z_j than if they were unequal. The full details of Z_j can be found in [SS99]. In this context it is sufficient to say that if $Z_j < 1$, the error of the combined strong classifier $H(x)$ is expected to decrease if the weak classifier $h(x)$ is chosen.

BoostMap is shown in algorithm 7. B is a set consisting of pairs (h_c, α_c) of classifiers and classifier weights. Lines 4 to 20 represent the weak learning steps of the algorithm. Lines 4 to 9 check if the Z_j value of any selected classifier, calculated with *inversed* classifier weight, is less than one. If so, it states that it is advantageous to add this classifier to B using its inversed weight. This is obviously the same as removing the classifier.

Lines 10 to 16 check all currently selected weak classifiers to see if there exists a new α which gives the classifier a Z_j value of less than one. If this is the case, it means that it is advantageous to add this classifier with the new weight, so the weight is effectively added to the classifier which is already included in B. A z value of 0.9999 is used in practice, to avoid many small weight adjustments which do not contribute significantly to the combined strong classifier. Note that the range of possible values for the new α is lower bounded by the existing weight. This is required to ensure that the embedding induced by the combined

Algorithm 7 BOOSTMAP

```
Input: $T = \{(x_1, y_1), ..., (x_t, y_t)\}$ , $|F_1|$, $|F_2|$          ▷ $T$ is the training set
 1: for all training triples $x_i$ do
 2:     $w_{i,1} \leftarrow 1/|T|$
 3: while termination conditions not met do
 4:     $z \leftarrow \min_{c=1,...|B|} Z_j(h_c, -\alpha_c)$
 5:     if $z < 1$ then
 6:         $g \leftarrow \text{argmin}_{c=1,...|B|} Z_j(h_c, -\alpha_c)$
 7:         $h_j, \alpha_j \leftarrow h_g, -\alpha_g$
 8:         $B \leftarrow B - (h_j, \alpha_j)$                              ▷ Remove classifier
 9:         goto 24
10:     $z \leftarrow \min_{c=1,...|B|}\{\min_{\alpha\in[-\alpha_c,\infty)} Z_j(h_c, \alpha)\}$
11:     if $z < .9999$ then
12:         $g \leftarrow \text{argmin}_{c=1,...|B|}\{\min_{\alpha\in[-\alpha_c,\infty)} Z_j(h_c, \alpha)\}$
13:         $h_j \leftarrow h_g$
14:         $\alpha_j \leftarrow \text{argmin}_{\alpha\in[-\alpha_c,\infty)} Z_j(h_g, \alpha)$
15:         $\alpha_g \leftarrow \alpha_g + \alpha_j$                              ▷ Modify weight
16:         goto 24
17:     $F_{j1} \leftarrow$ set of $|F_1|$ random $M1$ embeddings
18:     $F_{j2} \leftarrow$ set of $|F_2|$ random $M2$ embeddings
19:     $F_j \leftarrow F_{j1} \cup F_{j2}$
20:     $g \leftarrow \text{argmin}_{c=1,...|F_j|}\{\min_{\alpha\in[0,\infty)} Z_j(h_c, \alpha)\}$
21:     $h_j \leftarrow h_g$
22:     $\alpha_j \leftarrow \text{argmin}_{\alpha\in[0,\infty)} Z_j(h_c, \alpha)$
23:     $B \leftarrow B \cup (h_j, \alpha_j)$                                ▷ Add new classifier
24:     for all training triples $x_i$ do
25:         $w_{i,j+1} \leftarrow w_{i,j} \exp(-\alpha_j y_i h_j(x_i)) \,/\, Z_j(h_j, \alpha_j)$
26:     $j \leftarrow j + 1$
27: return $H(x) = \text{sign}(\sum_{j=1}^{J} \alpha_j h_j(x))$
```

classifier $H(x)$ is a metric space, because non-negative weights are required for non-negative distance values [AASK04b].

As for the problem of finding an optimal α value, it it convenient that Z_j is a convex function with exactly one extremal point [SS99]. Thus, $\min_{\alpha \in [-\alpha_c, \infty)} Z_j(h_c, \alpha)$ can be implemented as a binary search procedure, a strategy which has been followed in this work. The procedure simply starts at the existing weight value and doubles it until a point is reached where the gradient is positive. This point is then used as an upper bound, and a binary search is applied between the upper and lower bounds until the extremal point is found with the desired amount of precision.

The strong classifier $H(x)$ produced by BoostMap is a J-dimensional embedding, where J is the number of weak classifiers. The weights of the weak classifiers are used to define a weighted Manhattan distance. Given two objects u and v, the distance between them in the J-dimensional space can be calculated as shown in equation 3.6. The values u_j and v_j are found using the one-dimensional embedding associated with classifier h_j.

$$D((u_1, ..., u_J), (v_1, ..., v_J)) = \sum_{j=1}^{J} \alpha_j |u_j - v_j| \tag{3.6}$$

Given a non-metric distance function in the form of a $N \times N$ distance matrix, a straightforward way of deriving a metric distance matrix would be to use algorithm 7 directly. The metric distance matrix is then found by applying equation 3.6 to all $N \times N$ pairs of objects.

In the case of amino acid substitution matrices, however, we are dealing with *similarity* matrices. Recall that the one-dimensional $M1$ and $M2$ embeddings require *distances* between objects to be specified to be able to project them into one dimension. Therefore, it is necessary to convert the similarity matrix into a dissimilarity matrix before using BoostMap. Given a similarity matrix, the steps necessary to generate a distance matrix using BoostMap are:

1. Use equation 2.12 to convert the similarity matrix S into a distance matrix $D_{\text{non-metric}}$.
2. Use BoostMap to produce an embedding based on the distance matrix $D_{\text{non-metric}}$.
3. Obtain a metric distance matrix D_{metric} using the embedding produced by BoostMap and equation 3.6.

Readers should note that the presentation of the algorithm in [AASK04b] contains an error. Step 1 and 2 of the algorithm presented there states that a selected classifier should be removed if the Z_j value using its current classifier weight is less than one. The correct action is to remove a classifier if the Z_j value using its *inversed* current classifier weight is less than one, which has been confirmed by the authors.

3.5.3 Proposed BoostMap modifications

Recall that a BoostMap classifier checks whether j is closer to i than k is. In this work, such a classification in called a *triangular misclassification* if it does not correspond with the data in the original substitution matrix.

The strategy of minimizing an explicit measure of embedding quality (i.e. triangular misclassifications) separates BoostMap from many other embedding methods like classical multidimensional scaling and FastMap, which do not attempt to optimize a quantitative quality measure.

This seems especially appealing when the matrices are given in the form of similarity matrices, for example amino acid substitution matrices. Because the diagonal similarity scores of amino acid substitution matrices are not equal, there is no straightforward way of converting them into a distance matrix with $d(i,i) = 0$ for all $i \in \{1, ..., N\}$, metric or not, without losing information. As an example, consider applying the similarity to dissimilarity conversion defined in equation 2.12 to the PAM250 substitution matrix to obtain the distance matrix D_{PAM250}. Then check all triples (i, j, k) where $0 < i < j < k \leq 20$. For PAM250, check whether j is more similar to i than k is. For D_{PAM250}, check whether j is closer to i than k is. A simple program to do these checks reveals that the two matrices disagree on almost 20% of the triples.

This problem of transforming a similarity matrix (with unequal diagonal elements) into a distance matrix suggests that BoostMap may be a more appropriate choice than algorithms which try to minimize the distance between two matrices, because it is not clear which matrix we want to minimize the distance from. However, the straightforward way of using the BoostMap algorithm presented in the previous section has the paradoxical disadvantage of using a matrix where a relatively large amount of misclassifications has already been made. We are interested in a metric distance matrix which reflects the data in the similarity matrix as close as possible, not the intermediate distance matrix generated by equation 2.12.

To avoid this problem, two modifications to the algorithm are proposed for the purpose of handling similarity matrices like amino acid substitution matrices:

- Training triples $x = (i, j, k)$ are assigned labels based on the original similarity matrix, not the generated distance matrix. If j is more similar to i than k, x is assigned the label 1. If k is more similar to i than j, x is assigned the label -1. If j and k are equally similar to i, x is assigned the label 0.

- One-dimensional embeddings are extracted directly from the similarity matrix, in addition to embeddings from the distance matrix generated by equation 2.12.

The second point requires some further explanation. The one-dimensional embeddings produced by BoostMap are expected to maintain some degree of classification accuracy relative to the distance matrix generated by equation 2.12. However, it is not clear how accurate they will be relative to the original similarity matrix, since the distance matrix already includes a significant amount of classification error. In addition, a larger set of unique one-dimensional embeddings to choose from is intuitively better, since its greater degree of diversity increases the probability of finding a weak classifier which can correct the mistakes made by classifiers already chosen.

Therefore, a simple method of extracting one-dimensional embeddings from the similarity matrix is proposed. Assume that a reference object r is given. If the similarity matrix is symmetric, all similarities between r and other elements are given by a single column (or row) of the matrix ($s(r,t) = s(t,r)$ for all t because of symmetry). Thus, this column (or row) can be extracted and converted separately into a one-dimensional embedding. Converting this similarity vector sv_r into a distance vector is much simpler than converting the similarity matrix into a distance matrix. For amino acid substitution matrices, the largest element of such a similarity vector is $sv_r(r)$ in most cases. The corresponding distance vector dv_r can then be found as $dv_r(i) = sv_r(r) - sv_r(i)$ for all $i \in \{1, ..., N\}$. It represents a one-dimensional spatial embedding directly, since elements of the vector specifies the distance to the reference object r. Furthermore, this distance vector makes no misclassifications relative to the similarity vector from which it was generated. The name $M3$ will be used when referring to such embeddings below.

The case of two reference objects is a bit trickier. Using the same rationale as above, two columns (or rows) can be extracted from the similarity matrix. These need to converted into distance vectors so that the one-dimensional embedding can be computed using equation 3.4. However, a problem occurs if $sv_{x_1}(x_1) \neq sv_{x_2}(x_2)$. This means that the vector elements will be translated by different amounts when using the method described above for conversion into a distance vector. It is a problem because it leads to a situation where $dv_{x_1}(x_2) \neq dv_{x_2}(x_1)$, which in turn leads to ambiguity in relation to equation 3.4. An ad hoc solution which was found to work well is to calculate the mean of $sv_{x_1}(x_1)$ and $sv_{x_2}(x_2)$. The distance vector elements can then be found as shown in equation 3.7 for $j \in \{1, 2\}$ and $i \in \{1, ..., N\}$.

$$dv_{x_j}(i) = \frac{sv_{x_1}(x_1) + sv_{x_2}(x_2)}{2} - sv_{x_j}(i) \tag{3.7}$$

The resulting values of $dv_{x_1}(x_1)$ and $dv_{x_2}(x_2)$ are set to zero regardless of the result of the computation in equation 3.7. Using the two distance vectors dv_{x_1}

and dv_{x_2}, equation 3.4 is used to generate a one dimensional embedding. Embeddings generated in this way will be called $M4$ from here on.

In essence, the proposed modification of BoostMap can be summarized as follows:

1. Generate distance matrix S from similarity matrix $D_{\text{non-metric}}$ using equation 2.12.

2. Generate training set T using similarity matrix S to assign labels to triples of objects instead of distance matrix $D_{\text{non-metric}}$.

3. Run BoostMap as shown in algorithm 7, with the modification that $|F_3|$ $M3$ embeddings and $|F_4|$ $M4$ embeddings are generated in addition to the $M1$ and $M2$ embeddings. Use $D_{\text{non-metric}}$ to generate $M1$ and $M2$ embeddings. Use S to generate $M3$ and $M4$ embeddings. Set $F_j \leftarrow F_{j1} \cup F_{j2} \cup F_{j3} \cup F_{j4}$.

4. Obtain a metric distance matrix D_{metric} using the embedding produced by BoostMap and equation 3.6.

The inclusion of the $M3$ and $M4$ embeddings means that the computational complexity of BoostMap is increased. Although this is affordable in most cases when dealing with 20×20 matrices, the computation time can be controlled by tuning the number of embeddings considered at each iteration – obviously at the cost of less accuracy.

Source code for both the original and the modified versions of BoostMap is included on the enclosed CD.

3.6 Proposed genetic embedding algorithms

Two potential problems can be observed in the embedding algorithms presented in the previous sections:

- Algorithms FASTMAP and SIMILARITYNMDS use a fixed metric distance function. It is unclear whether this function is the best one to preserve the information in the original substitution matrix.

- Algorithms like FASTMAP and BOOSTMAP are based on the idea to project elements into spaces of lower dimensionality, and to combine these projections into a final embedding. This approach may not be able to represent all possible constellations of points in the d-dimensional space. In BOOSTMAP, each one-dimensional embedding represents one dimension of the d-dimensional embedding. Thus, the number of points which can be represented by the algorithm is determined by the number of possible one-dimensional embeddings.

To overcome these potential limitations, it is desirable to introduce higher degrees of freedom. An algorithm for embedding N points in d dimensions based on a (possibly) non-metric distance function should ideally have the following two degrees of freedom:

- The algorithm should be able to represent all possible constellations of points in d dimensions.
- The algorithm itself should choose the metric distance function which fits the data at hand best.

Higher degrees of freedom mean that the size of the solution space is increased dramatically, in turn leading to problems of higher computational complexity. The problem of finding an optimal spatial embedding with respect to some cost function can be seen as a search in a solution space consisting of all embeddings in a metric space. Removing the limitation of using a fixed distance function means that the size of the solution space is increased exponentially; it now consists of all embeddings in all metric spaces. An exhaustive search scheme is obviously not an option, since the size of the solution space is infinite. To find near-optimal candidate solutions to such problems, some kind of stochastic algorithm must therefore be applied. *Genetic algorithms* are popular stochastic algorithms for estimating near-optimal solutions to such problems.

In this section, two genetic algorithms for embedding N points in a metric space are proposed. First, an introduction to genetic algorithms is given by means of a preliminary attempt to use an evolutionary scheme to generate a metric distance function. Then a genetic algorithm which uses a fixed metric distance function is proposed. Finally, a *cooperative coevolutionary* genetic algorithm is proposed. This final algorithm eliminates the limitation of using a fixed metric distance function, and thus attempts to find a near-optimal metric embedding in a search space consisting of all possible embeddings in a large set of metric spaces.

3.6.1 A brief introduction to genetic algorithms

Most stochastic search algorithms operate on a single solution of the problem at hand. In contrast, genetic algorithms (often called *evolutionary algorithms*) operate on populations of many solutions from the search space. The idea is to evolve the population of solutions through a number of evolutionary steps which produce new *generations* of solutions. Each such step is designed so that it increases the average *fitness* of the candidate solutions in the population with respect to the problem. Fitness is simply the value of a function which estimates how capable the candidate solution is of solving the problem. For the problem of finding a metric distance matrix, a measure of metricity would obviously be included in

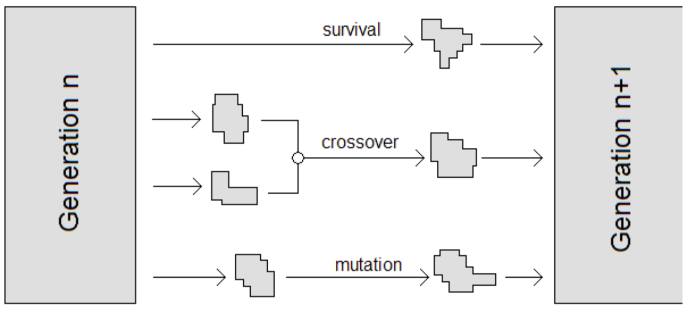


Figure 3.5: Evolutionary process of a generic genetic algorithm

the fitness function. Note that a *cost* function will be used in the genetic algorithms presented in this work. Obviously, maximizing a fitness function is the same as minimizing a cost function.

There are three basic steps in the evolutionary process of a genetic algorithm:

Survival Some of the fittest solutions survive by being copied directly from generation n to generation $n + 1$.

Crossover Two fit parent solutions are selected from generation n. A new solution is generated for generation $n + 1$ by applying a binary *crossover* operator to the two parent solutions. The crossover operator generates the new solution by copying some pieces from each of the parent solutions.

Mutation A small number of new solutions are generated for generation $n + 1$ by selecting a fit solution from generation n and applying a *mutation* operator to it. The mutation operator works by changing some pieces of the selected solution.

The same evolutionary process is then applied to the new generation of candidate solutions. Carefully designed genetic algorithms will guide the search into areas of the search space containing good candidate solutions to the problem at hand. A graphical illustration of the three basic steps of the evolutionary process is shown in figure 3.5, where candidate solutions are represented as geometrical constructions.

The search stops when the evolutionary process has reached a maximum number of generations, or when the fitness of the best solution found so far has reached an appropriate level. The search process is vulnerable to local extremal points because the crossover operator will typically produce candidate solutions which are similar to their parents. The purpose of the mutation operator is to introduce a jump in diversity in the population, thereby enabling the search process to escape such local extremes. This is illustrated in figure 3.6, where candidate solutions are shown as points of a cost function over the search space. Mutation allows one of the candidate solutions to escape the “valley” which all

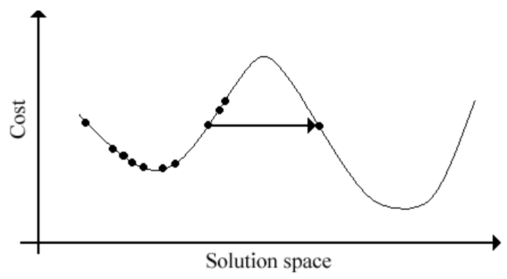


Figure 3.6: Mutation operator can guide the search out of local extremes

of the solutions are currently in. If this solution survives further evolutionary steps, a new sub-population can begin to evolve in the other "valley", which contains the global minimum. Note that the mutation step can be dropped if the fitness function contains no local maxima or, alternatively, if the cost function contains no local minima.

Readers interested in a more thorough introduction to genetic algorithms are referred to [BBM93a].

3.6.2 A preliminary genetic algorithm

As a preliminary solution to the problem of finding a metric distance matrix which corresponds as close as possible to the data in a non-metric substitution matrix, consider a solution space consisting of positive symmetric matrices. Algorithm 8 shows how the genetic approach could be used to find a metric distance matrix which retains some proximity to the original non-metric matrix. The initial population is generated by sampling values from a normal distribution (line 5). Elements from the non-metric distance matrix D are used as mean values, and input parameter σ is used as standard deviation. The parameter σ must be chosen with care; if the distance matrix D is far from being metric, it should be given a larger value than if D is almost metric.

Each evolutionary step starts by computing the cost of all matrices in the current population (line 12). In the same loop, each solution is checked to see if it is the best one found so far. If so, it is stored in a separate variable. Precomputing all cost values in this way is not strictly necessary. However, since each call to the TOURNAMENTSELECT procedure (to be explained below) implies checking the cost of several matrices from the population, this avoids having to recompute the cost of the same matrix more than once. The COST function is merely an implementation of equation 3.8. In this equation, $P[i]$ is the ith candidate solution matrix of the current population. $P[i]'_{m,n}$ is found using equation 2.22.

$$cost = \sum_{m=1}^{N-1} \sum_{n=m+1}^{N} \gamma(P[i]_{m,n} - D_{m,n})^2 + v(P[i]_{m,n} - P[i]'_{m,n}) \qquad (3.8)$$

Algorithm 8 Preliminary algorithm for generating metric distance matrices

```
Input: $D, S, \sigma, \alpha, \beta$        ▷ $D$ is an $N \times N$ matrix, $S$ is the population size
 1: $P \leftarrow \emptyset$        ▷ $P$ is the initial population
 2: while $|P| < S$ do
 3:     $C \leftarrow$ empty $N \times N$ matrix
 4:     for all upper triangular matrix elements $C_{m,n}$ do
 5:         $C_{m,n} \leftarrow \textsc{NormDist}(D_{m,n}, \sigma)$
 6:         $C_{n,m} \leftarrow C_{m,n}$
 7:     $P \leftarrow P \cup C$        ▷ Add matrix to initial population
 8: $\epsilon, B \leftarrow \infty, D$        ▷ $\epsilon$ is the cost of best solution $B$
 9: while $\epsilon$ is too large do
10:     $Q \leftarrow \emptyset$
11:     for $i \leftarrow 1, |P|$ do
12:         $E[i] \leftarrow \textsc{Cost}(P[i], D)$
13:         if $E[i] < \epsilon$ and $\textsc{IsMetric}(P[i])$ then
14:             $\epsilon, B \leftarrow E[i], P[i]$
15:     while $|Q| < \alpha|P|$ do        ▷ $\alpha$ controls survival rate
16:         $i \leftarrow \textsc{TournamentSelect}(E)$
17:         if $i$ has not been selected previously in this iteration then
18:             $Q \leftarrow Q \cup P[i]$
19:     $Q \leftarrow Q \cup \{\beta|P|$ elements crossed over from $P\}$
20:     $Q \leftarrow Q \cup \{(1 - \alpha - \beta)|P|$ elements mutated from $P\}$
21:     $P \leftarrow Q$
22: return $B$        ▷ Return the best solution found
```

As can be seen, the constant γ governs the influence of the distance between $P[i]$ and D. Likewise, the constant v governs the influence of the *non-metricity* of $P[i]$. Setting $\gamma = 0$ results in a cost function equal to the non-metricity value of the candidate solution matrix. In contrast, $v = 0$ results in a cost function equal to the squared distance between the candidate solution matrix and the original distance matrix. Obviously, the latter case is not interesting here, since we are interested in finding metric matrices.

Note that the call to the ISMETRIC function in algorithm 8 (line 13) can be skipped by having the COST function return two values, the squared distance between $P[i]$ and D and the non-metricity of $P[i]$. The call can then be replaced by a simple metricity value check.

Algorithm 9 TOURNAMENTSELECT

```
Input: $E$                                    ▷ $E$ is an array of cost values
Extern: $K$                         ▷ The value of $K$ is problem dependent
1: $min, k \leftarrow \infty, 1$
2: for $i \leftarrow 1, K$ do
3:     $j \leftarrow \lfloor$ RAND $\times |E| \rfloor + 1$
4:     if $E[j] < min$ then
5:         $min, k \leftarrow E[j], j$
6:     end if
7: end for
8: return $k$
```

Procedure TOURNAMENTSELECT generates K random indices for the list E of cost values, and returns the index corresponding to the least costly of the corresponding matrices. This is known as *tournament selection* (hence the name of the procedure), a selection technique commonly used in genetic algorithms. To avoid premature convergence in a local extremal point, we should not simply copy the $\alpha|P|$ best solutions from the current population. Parameter K allows us to adjust the balance between selection greediness and population diversity. Setting $K = |P|$ results in a greedy selection scheme. $K = 1$ is the same as selecting elements at random. The implementation is shown in algorithm 9, where RAND is a function which returns a number in the interval $[0, 1)$.

The crossover step (line 19) is not shown in detail. It simply selects two parent matrices from the current population by tournament selection. Upper triangular cells of the offspring matrix are then generated by selecting *k crossover points*, where the sequence of matrix cells between two crossover points are copied from a randomly selected parent. This can be achieved by storing the upper triangular cells of each matrix as a one-dimensional vector. For example, a one-point crossover starts by generating a random cut point. The offspring is then generated by copying the value on one side of the cut from one parent, and the value on the other side from the other parent. An alternative to this scheme is *uniform crossover*, where each matrix cell value is copied from a randomly selected parent. This is effectively the same as setting k equal to the number of upper triangular cells. Several analysis have been carried out to determine which technique is best, uniform crossover or k-point crossover. In [BBM93b], a summary of important comparison results is given. The conclusions are contradictive, and show that the choice of k is highly problem dependent.

The mutation step (line 20) picks a candidate solution matrix by tournament selection. This matrix is then mutated by sampling new matrix cell values from a normal distribution, where the original cell values are used as means and σ is used as standard deviation. As mentioned earlier, σ must be chosen large enough to allow the search process to escape local extremes.

In essence, algorithm 8 generates a matrix by minimizing a cost function which consists of two parts, the non-metricity of the matrix and the distance to the input matrix. This should guide the search into areas of the search space which contain candidate matrices of high metricity. The algorithm was, however, not found to perform well in practice. For the search to find any metric matrix at all, the non-metricity component of the cost must be weighted much heavier than the distance component. In addition, σ must be chosen so large that it causes the search to fluctuate out of areas where proximity to the original non-metric matrix is preserved.

The problem lies in the crossover operator of the algorithm. Given two parent matrices with some degree of metricity, how do we design a crossover operator which carries this metricity over to the child matrix? Observe that an $N \times N$ distance matrix can be seen as a fully connected graph of N nodes. A cut in this graph will affect all triangles which include edges crossing the cut. If there are N_1 nodes on one side of the cut and N_2 nodes on the other side, a total of $N_1 N_2$ triangles will be affected. Thus, the crossover operator presented above does not preserve the the triangle inequality well.

Nevertheless, algorithm 8 serves as an introduction to the idea of using a genetic scheme to generate a metric distance matrix. It is desirable to modify the algorithm so that metricity is *implicit* in the way the solutions to the problem are represented. The algorithm can then focus on preserving the information contained in the original non-metric matrix, and does not have to minimize metricity at the same time. The idea of embedding N elements in a metric space fits this requirement perfectly; rather than trying to find a metric distance matrix directly, a genetic scheme is used to find the least costly embedding. The next section presents the first of two such algorithms proposed in this work.

3.6.3 Algorithm using fixed metric distance function

The algorithm GAEMBEDDING (Genetic Algorithm Embedding) uses a fixed metric distance function to find a near-optimal embedding of N elements in d dimensions, in accordance with a specific cost function. Its pseudocode is shown in algorithm 10.

Algorithm 10 GAEMBEDDING

```
Input: $M, S, \sigma, \alpha, \beta, d$                         ▷ $M$ is a (possibly) non-metric matrix

 1: $R \leftarrow$ CLASSICALMDS($M$, $d$)                    ▷ $d$ is the number of dimensions
 2: $P \leftarrow \emptyset$
 3: while $|P| < S$ do                                ▷ $S$ is the size of populations
 4:     $C \leftarrow R$
 5:     for each matrix element $C_{m,n}$ do                 ▷ Generate initial population
 6:         $C_{m,n} \leftarrow$ NORMDIST($C_{m,n}$, $\sigma$)        ▷ $\sigma$ controls population diversity
 7:     $P \leftarrow P \cup C$

 8: $\epsilon$, $B \leftarrow \infty$, $R$                              ▷ $\epsilon$ is cost of best solution $B$
 9: while $\epsilon$ is too large do
10:     $Q \leftarrow \emptyset$
11:     for $i \leftarrow 1, |P|$ do
12:         $E[i] \leftarrow$ GAEMBEDDINGCOST($P[i]$, $M$)
13:         if $E[i] < \epsilon$ then
14:             $\epsilon, B \leftarrow E[i], P[i]$
15:     while $|Q| < \alpha|P|$ do                           ▷ $\alpha$ controls survival rate
16:         $i \leftarrow$ TOURNAMENTSELECT($E$)
17:         $Q \leftarrow Q \cup P[i]$
18:     while $|Q| < (\alpha + \beta)|P|$ do                    ▷ $\beta$ controls crossover rate
19:         $i \leftarrow$ TOURNAMENTSELECT($E$)
20:         $j \leftarrow$ TOURNAMENTSELECT($E$)
21:         $C \leftarrow$ empty $N \times d$ matrix
22:         for each matrix element $C_{m,n}$ do
23:             if RAND $< .5$ then
24:                 $C_{m,n} \leftarrow P[i]_{m,n}$               ▷ Select from mother embedding
25:             else
26:                 $C_{m,n} \leftarrow P[j]_{m,n}$                ▷ Select from father embedding
27:         $Q \leftarrow Q \cup C$
28:     while $|Q| < |P|$ do                          ▷ Mutation rate is $1 - \alpha - \beta$
29:         $i \leftarrow$ TOURNAMENTSELECT($E$)
30:         $C \leftarrow P[i]$
31:         for each matrix element $C_{m,n}$ do
32:             if RAND$< .5$ then
33:                 $C_{m,n} \leftarrow$ NORMDIST($C_{m,n}$, $\sigma$)
34:         $Q \leftarrow Q \cup C$
35:     $P \leftarrow Q$                          ▷ Update population for next iteration

36: for each row vector $B_i$ of $B$ do
37:     for each row vector $B_j$ of $B$ do
38:         $O_{i,j} = \sqrt{(B_i - B_j)(B_i - B_j)^{\mathrm{T}}}$                ▷ Euclidean distance
39: return $O$                                    ▷ Return metric distance matrix
```

Line 1 of the algorithm uses classical metric multidimensional scaling to obtain an initial candidate embedding of the N elements in d dimensions. As explained in section 3.3, metric multidimensional scaling could be problematic if the data at hand is not metric. Nevertheless, it quickly produces an embedding which can be seen as an approximate solution. An alternative would be to generate random constellations of N points in d dimensions. Some initial trials showed that the two alternatives were able to find embeddings of near equal minimal cost as long as σ was chosen large enough, but typically after a smaller number of iterations when starting with the metric MDS solution.

Lines 5 to 7 generate an initial population of embeddings. Each embedding C is coded as a $N \times d$ matrix, where each row corresponds to the coordinates of one element. This is the same way of representing embeddings as the X matrix used by FastMap (see figure 3.3). Coordinates are sampled from a normal distribution where the corresponding coordinate of the metric MDS solution is used as mean and the user defined parameter σ as standard deviation.

At each iteration of the algorithm, the next generation population Q is initialized as an empty set. The costs of all embeddings in the current population are then calculated using the GAEMBEDDINGCOST procedure, which is explained below. Each embedding is checked to see if it is the best one found so far. Then a total of $\alpha|P|$ embeddings are copied directly from population P to population Q in lines 15 to 17. Each surviving embedding is picked by tournament selection.

A total of $\beta|P|$ new embeddings are generated by crossover as follows. A father and a mother embedding is picked by tournament selection. For each coordinate $n \in \{1, ...d\}$ of each element $m \in \{1, ..., N\}$, a uniformly distributed random number from the interval $[0, 1]$ is sampled. If this number is lower than 0.5, the coordinate is copied from the father embedding. If not, it is copied from the mother embedding.

Finally, $(1 - \alpha - \beta)|P|$ elements of Q are generated in lines 28 to 34 by "mutating" elements picked by tournament selection from P. The mutation is achieved by sampling new coordinate values from a normal distribution, using the original coordinate value as mean. This is a rather radical kind of mutation. In fact, it does not correspond with the biological meaning of the word at all. In most kind of genetic algorithms where individuals from a population are represented as bit vectors, the mutation step is achieved by flipping the bit value in one position of such a vector. However, the steps used in lines 28 to 34 of GAEMBEDDING were found to work well in practice. Observe that embeddings are "mutated" in the same way as the initial population is generated. If the embedding picked by tournament selection is close to a local extremal point, sampling a new embedding using the same measure of diversity as was used to generate the initial population should help to guide the search out of this local extreme. Changing one coordinate of one embedded element represents a much smaller spatial change, and is intuitively not very likely to guide the search out of the local minimum.

It is assumed that the size of the population remains the same between iterations. Thus, we require $\alpha + \beta \leq 1$. If $\alpha + \beta = 1$, the entire mutation step will be skipped. Typical values found to work well are $\alpha = 0.1$ and $\beta = 0.88$, which means that the mutation rate is 0.02.

The cost of an embedding is computed as shown in algorithm 11. In addition to the embedding, it takes the original similarity matrix M as an input parameter. M is first converted into a distance matrix D using equation 2.12. In an implementation, this would of course be done once at the start of GAEMBEDDING, and D would be passed as an extra parameter to GAEMBEDDINGCOST. This has been skipped in the pseudocode for clarity and simplicity. Using the coordinates of the N embedded elements, represented as rows of C, euclidean distances are then calculated between points. The squared sum of differences between the two distance matrices D and O is stored in the variable δ.

Algorithm 11 GAEMBEDDINGCOST

```
Input: $C, M$          ▷ $C$ is an embedding, $M$ is an $N \times N$ similarity matrix
Extern: $\varphi, \theta$          ▷ Scaling constants
 1: $\delta, m \leftarrow 0, 0$
 2: $D \leftarrow$ distance matrix from using eq. 2.12 on $M$
 3: for each row vector $C_i$ of $C$ do
 4:     for each row vector $C_j$ of $C$ do
 5:         $O_{i,j} = \sqrt{(C_i - C_j)(C_i - C_j)^{\mathrm{T}}}$          ▷ Euclidean distance
 6:         $\delta \leftarrow \delta + (D_{i,j} - O_{i,j})^2$
 7: for each triple $(i, j, k)$ where $0 < i < j < k \leq N$ do
 8:     $a \leftarrow M_{i,k} - M_{i,j}$
 9:     $b \leftarrow O_{i,j} - O_{i,k}$
10:     if ($b > 0$ and $a \leq 0$) or ($a > 0$ and $b \leq 0$) then
11:         $m \leftarrow m + 1$
12: return $\varphi m + \theta\delta$
```

Lines 7 to 11 of the algorithm are essentially adopted from BoostMap. Using M, all unique triangles (i, j, k) of distinct elements are checked to see whether j is more similar to i than k is. Similarly, O is used to check whether j is closer to i than k is in the embedding. If the two tests disagree, m is incremented.

The final cost of the embedding is returned as a weighted sum of δ and m. The constant φ is typically chosen to be larger than θ. In fact, some of the best results in this work were obtained by setting $\theta = 0$. In that case, GAEMBEDDING essentially performs the same minimization as BoostMap. They still differ in methodology and distance function, though.

Note that euclidean distances between points are used both in GAEMBEDDINGCOST and in GAEMBEDDING to obtain the final metric distance matrix. Any other metric distance function could be chosen. This freedom to plug any metric distance function into the genetic embedding algorithm was the main inspiration behind the cooperative coevolutionary algorithm proposed in the next section.

3.6.4 Algorithm using evolving metric distance function

This section proposes a genetic algorithm that minimizes the cost function with respect to both spatial embedding and distance function. Some background theory is given before the algorithm itself is presented.

If an embedding algorithm is to learn a metric distance function itself, it must basically learn two things at once. Observe that the problems of finding the least costly embedding and finding the least costly distance function are intertwined; they both strive to minimize the same cost function. There should be some sort of *cooperation* between the process of minimizing the cost function with respect to embeddings and the process of minimizing the cost function with respect to distance function. Two main questions must be answered to be able to construct an algorithm which achieves this:

- How can such a cooperation scheme be incorporated into a genetic algorithm?
- How do we represent a distance function as a data structure in a genetic algorithm?

The first question will be addressed first. Using a genetic scheme, a population of distance functions should be evolved in parallel with the population of spatial embeddings. Genetic algorithms for evolving populations in parallel are called *coevolutionary* algorithms [Par98]. There are two basic types of coevolutionary genetic algorithms, *competitive* and *cooperative* ones [WLDJ01]. Competitive coevolutionary algorithms evolve different populations by having them compete against each other. The fittest individuals are the ones which are most successful in such competitions. In cooperative coevolutionary algorithms, on the other hand, different populations represent different parts of the problem to be solved. A candidate solution consists of one individual from each population. The populations are cooperating in the sense that they all strive to minimize the same cost function (or maximize the same fitness function). It is clear that a cooperative coevolutionary scheme is the most natural one in this case. We have two populations, each of which contains parts of candidate solutions. A candidate solution is merely a pair of a spatial embedding and a metric distance function.

The basic structure of a cooperative coevolutionary scheme is shown in algorithm 12. As can be seen, the coevolution lies in the computation of cost values. A central question is: How do we assign a cost value to an individual? Since a complete candidate solution is required to calculate a cost value, the cost of an

Algorithm 12 Basic structure of a cooperative coevolutionary algorithm

```
1: Initialize all populations
2: while not terminated do
3:     for all populations do
4:         for each individual $i$ in population do
5:             Select collaborators from all other populations
6:             Calculate cost of $i$ based on encounters with collaborators
7:         Select individuals for survival
8:         Select individuals by crossover
9:         Select individuals by mutation
10:        Update population
```

individual in a population is highly dependent on which individuals we select from other populations. Calculating the average cost value is, in most cases, too computationally expensive to be an option. Only a few individuals can be selected from each population. This problem of selecting collaborators from other populations has been analysed empirically in [WLDJ01]. Questions addressed include:

- What degree of greediness should be used when selecting collaborators? Should the best individuals from the other populations be selected, or should individuals from other populations be selected randomly?
- Given that many collaborators are selected from each of the other populations, which cost value should be chosen? An optimistic approach would be to choose the smallest cost observed. A pessimistic approach would be to choose the highest cost observed.

The analysis concludes that an optimistic approach is generally best for choosing cost value. The degree of greediness, however, is more problem dependent.

The second initial question remains to be answered: How do we represent a distance function as a data structure in a genetic algorithm? Given two points x and y with position row vectors $\mathbf{x}$ and $\mathbf{y}$, first note that the euclidean distance between them is found as $d(x,y) = \sqrt{(\mathbf{x}-\mathbf{y})(\mathbf{x}-\mathbf{y})^{\mathrm{T}}} = \sqrt{(\mathbf{x}-\mathbf{y})I(\mathbf{x}-\mathbf{y})^{\mathrm{T}}}$, where I is the identity matrix. If I is replaced with a matrix A whose inverse matrix has the property of a covariance matrix, we get a distance function known as the *Mahalanobis* distance between x and y. This distance function is shown in equation 3.9.

$$d(x,y) = \sqrt{(\mathbf{x}-\mathbf{y})A(\mathbf{x}-\mathbf{y})^{\mathrm{T}}} \tag{3.9}$$

In statistics, the Mahalanobis distance is the dissimilarity measure between two random vectors $\mathbf{x}$ and $\mathbf{y}$ (from the same distribution) with covariance matrix

A^{-1}. Thus, the difference between the euclidean distance and the Mahalanobis distance is that the latter takes correlations of the data set into account. To ensure metricity, A must be positive semi-definite, i.e. it must satisfy $\mathbf{x}A\mathbf{x}^{\mathrm{T}} \geq 0$ for all $\mathbf{x}$ [XNJR03]. If A is restricted to be a matrix with non-negative values on the diagonal and zeros elsewhere, we obtain a distance function where different axes of the d-dimensional space are weighted by different amounts. Such a matrix is clearly positive semi-definite, since the product $\mathbf{x}A\mathbf{x}^{\mathrm{T}}$ cannot possibly be negative if the elements of A are non-negative on the diagonal and zero elsewhere.

Clearly, a Mahalanobis distance function can be represented by its A matrix. The core idea of the proposed coevolutionary algorithm is to evolve a population of such matrices in parallel with the population of spatial embeddings. The matrices are restricted to have non-zero values on the diagonal and zeros elsewhere. This is because the general case of positive semi-definite matrices causes problems when generating new matrices by crossover and mutation; it is hard to find operators of low computational complexity which preserve the properties of positive semi-definite matrices. In contrast, this is easy to ensure if the Mahalanobis matrices are restricted to be diagonal.

Algorithm 13 shows CCAEMBEDDING, the proposed method of embedding N points using a coevolutionary scheme. Lines 3 to 12 initialize the two populations P_1 and P_2. The population of $N \times d$ embeddings, P_1, is initialized as in GAEMBEDDING. The population of $d \times d$ Mahalanobis matrices, P_2, is initialized by sampling diagonal elements from a normal distribution with the parameters μ and σ_2 as mean and standard deviation.

The algorithm then follows a standard coevolutionary scheme. Procedure CCAEMBEDDINGCOST, which is explained in detail below, is used to calculate the cost of individual embeddings and matrices. When calculating the cost of an embedding, the population of matrices must be supplied as an argument to the procedure. This is because collaborators will have to be selected to be able to estimate the cost. Similarly, the population of embeddings must be provided as an argument when calculating the cost of a matrix.

Lines 25 to 27 follow the exact same steps as the crossover and mutation steps of GAEMBEDDING. The same operators are also applied to the population of Mahalanobis matrices in lines 27 to 30, with two differences:

- The operators are applied to $d \times d$ matrices instead of $N \times d$ matrices (embeddings).
- The mutation operator must assure that no matrix elements are negative.

Finally, lines 33 to 35 calculate the distances between embedded points using the spatial embedding B and the Mahalanobis matrix A, which were found to minimize the cost. The properties of A ensure that the resulting distance matrix is metric.

Algorithm 13 CCAEMBEDDING

```
Input: $M, S_1, S_2\ \sigma_1, \sigma_2, \mu, \alpha, \beta, d$
Extern: CCAEMBEDDINGCOST

 1: $R \leftarrow$ CLASSICALMDS($M, d$)
 2: $P_1, P_2 \leftarrow \emptyset, \emptyset$
 3: while $|P_1| < S_1$ do
 4:     $C \leftarrow R$
 5:     for each matrix element $C_{m,n}$ do
 6:         $C_{m,n} \leftarrow$ NORMDIST($C_{m,n}, \sigma_1$)
 7:     $P_1 \leftarrow P_1 \cup C$
 8: while $|P_1| < S_2$ do
 9:     $F \leftarrow$ empty $d \times d$ matrix
10:     for each diagonal matrix element $F_{m,m}$ do
11:         $F_{m,m} \leftarrow \max(0,$ NORMDIST$(\mu, \sigma_2))$
12:     $P_2 \leftarrow P_2 \cup F$

13: $\epsilon, B \leftarrow \infty, R$                        ▷ $\epsilon$ is cost of best embedding $B$
14: $A \leftarrow d \times d$ identity matrix                 ▷ $A$ is the best parameter matrix
15: while $\epsilon$ is too large do
16:     $Q_1, Q_2 \leftarrow \emptyset, \emptyset$
17:     for $i \leftarrow 1, |P_1|$ do
18:         $E_1[i], F \leftarrow$ CCAEMBEDDINGCOST($P_1[i], P_2, M$)
19:         if $E_1[i] < \epsilon$ then
20:             $\epsilon, B, A \leftarrow E_1[i], P_1[i], F$
21:     for $i \leftarrow 1, |P_1|$ do
22:         $E_2[i], F \leftarrow$ CCAEMBEDDINGCOST($P_2[i], P_1, M$)
23:         if $E[i] < \epsilon$ then
24:             $\epsilon, B, A \leftarrow E_2[i], F, P_2[i]$
25:     $Q_1 \leftarrow Q_1 \cup \{\alpha|P_1|$ tournament selected embeddings from $P_1\}$
26:     $Q_1 \leftarrow Q_1 \cup \{\beta|P_1|$ embeddings crossed over from $P_1\}$
27:     $Q_1 \leftarrow Q_1 \cup \{(1-\alpha-\beta)|P_1|$ embeddings mutated from $P_1\}$
28:     $Q_2 \leftarrow Q_2 \cup \{\alpha|P_2|$ tournament selected matrices from $P_2\}$
29:     $Q_2 \leftarrow Q_2 \cup \{\beta|P_2|$ matrices crossed over from $P_2\}$
30:     $Q_2 \leftarrow Q_2 \cup \{(1-\alpha-\beta)|P_2|$ matrices mutated from $P_2\}$
31:     $P_1 \leftarrow Q_1$
32:     $P_2 \leftarrow Q_2$

33: for each row vector $B_i$ of $B$ do
34:     for each row vector $B_j$ of $B$ do
35:         $O_{i,j} = \sqrt{(B_i - B_j)A(B_i - B_j)^{\mathrm{T}}}$
36: return $O$                                           ▷ Return metric distance matrix
```

Algorithm 14 CCAEMBEDDINGCOST

```
Input: $X, Y, M$
Extern: $K, \varphi, \theta$
 1: $k \leftarrow 1$ if $X$ is an embedding, 0 if not
 2: $D \leftarrow$ distance matrix from using eq. 2.12 on $M$
 3: $\varepsilon, W \leftarrow \infty, []$
 4: for $l \leftarrow 1, K$ do
 5:     $n \leftarrow \lfloor \text{RAND} \times |Y| \rfloor + 1$
 6:     $Z \leftarrow Y[n]$
 7:     if $k = 0$ then
 8:         $T \leftarrow Z$
 9:         $Z, X \leftarrow X, T$
10:     $\delta, m \leftarrow 0, 0$
11:     for each row vector $X_i$ of $X$ do
12:         for each row vector $X_j$ of $X$ do
13:             $O_{i,j} = \sqrt{(X_i - X_j)Z(X_i - X_j)^{\mathrm{T}}}$
14:             $\delta \leftarrow \delta + (D_{i,j} - O_{i,j})^2$
15:     for each triple $(i, j, k)$ where $0 < i < j < k \leq N$ do
16:         $a \leftarrow M_{i,k} - M_{i,j}$
17:         $b \leftarrow O_{i,j} - O_{i,k}$
18:         if ($b > 0$ and $a \leq 0$) or ($a > 0$ and $b \leq 0$) then
19:             $m \leftarrow m + 1$
20:     if $(\varphi m + \theta\delta) < \varepsilon$ then
21:         $\varepsilon \leftarrow (\varphi m + \theta\delta)$
22:         if $k = 0$ then
23:             $W \leftarrow X$
24:         else
25:             $W \leftarrow Z$
26: return $\varepsilon, W$
```

Algorithm 14 shows CCAEMBEDDINGCOST, the procedure used to estimate the cost of an individual from one of the two populations. Given an embedding or a Mahalanobis matrix, the idea is to pick K random individuals from the other population. The lowest observed cost is then returned. This is an optimistic approach to the problem of cost assignment. The greediness is, however, not as high as possible; observe that none of the collaborators are selected based on cost values from the previous evolutionary round. Instead, an approach based on tournament selection has been adopted. Preliminary tests indicated that this approach seems to be better than a greedy approach where the highest ranked individuals from the previous round are chosen.

Lines 7 to 9 and 22 to 25 are needed to determine the meaning of input parameters and variables. If X is a an embedding, then Y is the population of Mahalanobis matrices. Similarly, if X is a Mahalanobis matrix, then Y is the population of embeddings. For the core cost calculations in steps 10 to 19 to make sense, X and Z must therefore be swapped (in lines 7 to 9) if X is not an embedding. It is simply a way of enabling the same procedure to be used for both populations.

The core cost calculation steps of the procedure, lines 10 to 19, are exactly the same as the cost calculation steps in GAEMBEDDINGCOST (algorithm 11). In other words, CCAEMBEDDING minimizes a weighted sum of the number of misclassifications and the squared distance between an embedding and an approximate distance matrix.

It should be noted that the introduction of an additional population increases the computational demands of the algorithm significantly. In general, the population sizes of CCAEMBEDDING must be chosen to be smaller than the population size of GAEMBEDDING. The parameter K in CCAEMBEDDINGCOST is also crucial to the performance of the algorithm, since it determines how many times the core cost calculation loop must be run for each individual in each population in each iteration. Promising results have been obtained when setting $K = 4$.

The GAEMBEDDING and CCAEMBEDDING algorithms proposed in this section have been implemented in Matlab. Although Matlab is an interpreted and untyped programming language, acceptable performance is achieved because many of the calculations can be expressed as matrix operations. The source code is included on the enclosed CD.

3.7 Summary of proposed methods and algorithms

Tom sum up the methods and algorithms presented in this chapter, two main methods have been identified and presented:

- The triangle fixing algorithm solves the problem of making a non-metric distance matrix metric by enforcing triangle constraints until all of them are satisfied.

- The embedding algorithms solve the problem by mapping an $N \times N$ substitution matrix into a metric space. Thereafter, a metric distance matrix is recovered using the distance function of the metric space.

The embedding approach requires a concrete algorithm to be chosen. Five different algorithms are considered in this work:

NMDS attempts to find an optimal metric embedding in the sense that the difference in rank between inter-object distances and similarity values from the original substitution matrix is minimized. Some small modifications of the original algorithm have been proposed and included in the implementation.

FastMap assumes that the substitution matrix values are inter-object distances in a space of a certain dimensionality. Coordinates are then found by recursively projecting objects into hyperplanes of one less dimension. No explicit measures of embedding quality are minimized.

BoostMap combines many one-dimensional embeddings into a high-dimensional embedding using adaptive boosting. The algorithm attempts to minimize the number of triangular misclassifications. Modifications of the original algorithm have been proposed and included in the implementation.

GAEmbedding evolves a population of embeddings of fixed dimensionality by attempting to minimize a cost function. The cost function is a weighted sum of the squared distance to the original substitution matrix and the number of triangular misclassifications.

CCAEmbedding uses a coevolutionary scheme with one population of spatial embeddings and one population of Mahalanobis matrices. The cost function is the same as the single-population genetic algorithm.

An analytical comparison of the algorithms in terms of computational complexity is given in chapter 5.

Having identified a set of possible methods for solving the problem, we are naturally interested in knowing which ones perform best in terms of begin able to retain the information inherent in the original matrix. Are there any differences between the algorithms? If so, which factors determine these differences? Is the triangle fixing approach better than the embedding approach? To answer these question, this work compares the methods using empirical experiments. The experiments and the results obtained from them are the subjects of chapter 4.

Chapter 4

Experiments and results

The algorithms presented in chapter 3 are general in the sense that any matrix can be used as input to produce a metric distance matrix. However, the non-metricity of amino acid substitution matrices in the field of bioinformatics has been the main motivation for the development of the methods. As mentioned in section 2.5.5, finding information-preserving metric mappings for such matrices are highly desirable, since it enables data to be indexed efficiently. Much research effort has therefore been put into finding metric conversions for such substitution matrices. As they have been the main motivation behind this work, they have also been chosen as subjects of evaluation here.

Chapter 2 explained how amino acid substitution matrices are used for homology search in databases of proteins. When measuring the performance of such matrices, we are mainly interested in their sensitivity, i.e. their ability to find the biologically correct homologous proteins. This should be reflected in the chosen method of testing and evaluating the proposed algorithms. The algorithms differ in the way they attempt to preserve the information inherent in the original substitution matrix. The experiments must therefore try to uncover which algorithm uses the “correct” way of preserving the information, in the sense that the sensitivity of the input matrix is preserved as good as possible.

This chapter presents the experiments which have been carried out for the purpose of testing the performance of the proposed methods and algorithms. Section 4.1 presents the methods which have been chosen for evaluating the matrices, and the dataset which has been used. Section 4.2 presents the actual results. Finally, section 4.3 performs tests for statistical significance on the results.

4.1 Evaluating generated substitution matrices

To be able to assess the performance of the metric matrices produced by the proposed methods, it is quite clear that they should be used as substitution matrices in a homology search. The results from such a search can then be compared with the results from a search with the original non-metric matrix. This section explains in detail how this has been done, by presenting the method of evaluation and the dataset which has used. A final section gives a description of the scheme used to measure the sensitivity of the matrices.

4.1.1 Method of evaluation

To compare the generated substitution matrices with the original ones, the Smith-Waterman algorithm for computing local alignments, presented in section 2.2.3, has been implemented. Gotoh's optimizations for affine gap penalty have been included. Search results obtained using the generated substitution matrices can then be compared to search results obtained using the original substitution matrices. To get the true optimal alignment scores, a straightforward implementation using no heuristics was developed. The `SSEARCH` program from the FASTA[1] package was considered, but ultimately discarded because it includes heuristics to achieve some computation speed improvements. In addition, the programs in the FASTA package do not allow custom substitution matrices to be specified.

Because the Smith-Waterman algorithm uses *similarity scores* to compute alignments, the dissimilarity values in the metric matrices must be converted into similarity values before the algorithm can be applied. The version of the algorithm presented in [Got82] uses a matrix of dissimilarity values, but proves to be problematic to use for computing local alignments. When computing the score using similarity scores, observe that a local alignment can immediately be closed if its similarity value drops below zero. In contrast, the version using distance penalties will have to specify a maximal acceptable distance value beyond which further expansion of the alignment can be skipped. Such a maximal distance value is not available in most cases.

Therefore, the strategy of converting the metric distance matrices into similarity matrices has been chosen in this work. This can be achieved by simply subtracting the maximum matrix value from the distance value in a matrix cell, which produces a similarity value. However, the correctness of the Smith-Waterman algorithm requires a mix of positive and negative similarity scores in the substitution matrix. Xu et al. has found in [XM04] that the best way of achieving this is to subtract the median matrix element from the positive similarity matrix generated as described above. This strategy has also been adopted in this work. The resulting similarity matrix is also scaled by a constant which minimizes the squared distance to the original matrix. Note that these transformations do not pose any problems to the algorithms which are based on preserving triangular proximities; if j is classified as being closer to i than k is, then $y(j + x)$ will

[1]http://fasta.bioch.virginia.edu/

be classified as being closer to $y(i + x)$ than $y(k + x)$ will. In other words, the proximities are invariant to translation and scaling.

A program was developed for sequentially searching a database of proteins. The program reads a sequence of query proteins in FASTA format. For each of these proteins, the entire database is searched sequentially for homology. Similarity between a query protein and a database protein is measured using the Smith-Waterman local alignment score, which is computed using the input substitution matrix.

As may be apparent, this brute force way of computing local alignments and sequentially searching the entire database is computationally heavy. In this context, however, we are interested in obtaining the most accurate measure of the sensitivity of the substitution matrices. Thus, exact search results are more interesting than heuristic search results obtained using algorithms such as BLAST and FASTA.

The implementation of the database search program allows users to define subsets of the set of query proteins. Parallel processing is thereby enabled – simply by running the program with different subsequences of queries on different computers. Using the test database presented in section 4.1.2, this enables the entire search process to be completed in reasonable time (in order of minutes) using a couple of workstations.

The C++ source code for the search program can be found on the enclosed CD. The CD also contains Matlab source code (which is compatible with GNU Octave[2]) for the methods and algorithms presented in chapter 3.

4.1.2 Dataset

To test the sensitivity of the generated substitution matrices, three types of data sets are required:

- A database of proteins coded as sequences of amino acid symbols.
- A set of query sequences to be used when searching the database for homology.
- For each query sequence, a list of true positive hits in the database.

In this work, the database used in both [SAM$^+$01] and [XM04] has been adopted. This database consists of 6433 proteins from *saccharomyces cerevisiae*, commonly known as *baker's* or *budding* yeast. The query set contains a total of 103 sequences. Accompanying these queries are 103 distinct lists of true database hits. These true hits have been identified by human domain experts. All data

[2]http://www.octave.org/

is publically available from NCBI (National Center for Biotechnology Information) at `ftp://ftp.ncbi.nlm.nih.gov/pub/impala/blasttest/`. The data used in this work were downloaded in February 2005.

Some proteins contain the symbols *B*, *Z* and *X*, corresponding to amino acids which have not been uniquely determined. Seven of the 103 query sequences contain such symbols: *ACTIN*, *ARR*, *CATH*, *DHHC*, *DNASE1*, *HISDAC* and *S1*. Some of the substitution matrices to be tested only specify scores for the 20 amino acids, not for these additional symbols. This is a problem, since it is desirable to be able to compare results from different substitution matrices. A choice was therefore made to remove these seven sequences from the query set. Thus, the query set used in this work consists of 96 query sequences.

4.1.3 Performance measure

When measuring the performance of homology search results, a modified version of *receiver operating characteristics* (ROC) scoring introduced by Gribskov and Robinson in [GR96] has established itself as a common choice in the field of bioinformatics. Given a ranked list of search results and a list of true positive hits, the ROC_n value is calculated as shown in equation 4.1.

$$\text{ROC}_n = \frac{1}{nT}\sum_{i=0}^{n} t_i \tag{4.1}$$

In this equation, T is the total number of true positives, and t_i is the number of true positive hits ranked ahead of the ith false positive. Obviously, given a list of true positive hits of length m, one needs a search result list of length $n+m$ to compute the ROC_n score. A score of 1 indicates a perfect match, where the first m elements of the result list are true positive hits. A score of 0 indicates that no true positive hits were found among the n highest ranked elements of the result list.

In bioinformatics, the ROC_{50} score has been chosen as a performance measurement in many recent works where homology search results are evaluated. As noted in [GB99], "this value has the advantages of yielding a wider spread of values, requiring less storage space, and corresponding to the typical biologist's willingness to sift through only about 50 false positives". The ROC_{50} score is used a number of places referenced from this work. In [XM04], ROC_{50} is used to measure the performance of the metric mPAM matrix (see section 2.6). It is also used to evaluate the search results in [HHLB04] and [GB99].

ROC_{50} has also been chosen to measure the performance of the matrices in this work, mainly motivated by its widespread usage in similar works. This makes the results from the methods proposed here comparable to other results where ROC_{50} has been used. In particular, it enables the results to be directly compared to the results presented in [XM04], which uses the same dataset as this work.

Generated substitution matrices can then be evaluated by comparing the ROC_{50} score obtained from a database search with the ROC_{50} score from a search with the original matrix.

4.1.4 Matrices

A representative selection of commonly used substitution matrices must be selected as subjects of evaluation. The PAM and BLOSUM family of substitution matrices are by far most commonly used. Different members of these families represent different levels of evolutionary divergence. Choosing a matrix therefore depends on the expected degree of sequence divergence in the data at hand. This is illustrated in figure 4.1.

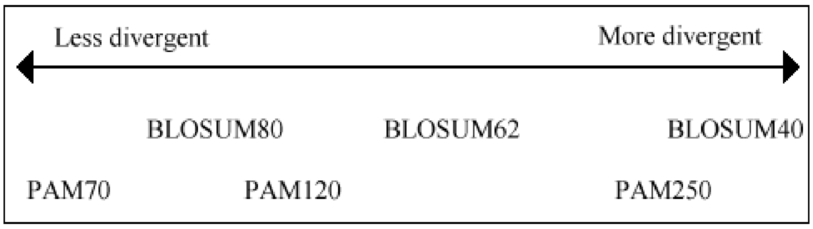


Figure 4.1: Relation between sequence divergence and substitution matrices

Thus, to evaluate methods across the scale of divergence, the following matrices have been selected:

- PAM70, PAM120, PAM250
- BLOSUM80, BLOSUM62, BLOSUM40

Each of the algorithms presented in chapter 3 are applied to each of these matrices. The resulting metric distance matrices are then supplied as inputs to the database search program described in section 4.1.1. In addition, the mPAM250 matrix introduced in section 2.6 has been included in the evaluation of metric distance matrices derived from the PAM250 matrix. Since 6 proteins have been removed from the query set in this work, the ROC_{50} performance score has been recalculated for the mPAM250 matrix rather than using the score presented in the original paper.

4.2 Homology search results

The homology search results presented in this section were obtained by running the database search program in parallel on three different computers – one running Linux 2.4.30 on an x86/32 processor, a second one running Linux 2.6.11-1 on an x86/64 processor and a third one running Solaris 9 on an ultrasparc processor. The results were then gathered from the three computers and average ROC$_{50}$ scores were calculated. Metric distance matrices were derived using the Matlab implementations of the algorithms. All the produced algorithms are included on the enclosed CD. For each algorithm, parameters were tuned by repeatedly generating new distance matrices and running trial searches on the database. The parameters are presented in the list below, along with abbreviation chosen for the algorithms (to simplify notation in tables and figures).

- **TF** - Triangle Fixing algorithm, presented in section 3.1. The value of the constant ϵ was set to 0.0001, which enabled the algorithm to converge in a metric distance matrix.
- **NMDS** - Non-metric Multidimensional Scaling, presented with proposed modifications in section 3.3.2. The value of the constant s was set to $0.1 \times N^{-3}$, in accordance with the advice in [TO04]. The maximum number of iterations was set to 5000, and the number of dimensions was set to 19.
- **FM** - FastMap, presented in section 3.4. The number of dimensions was set to 19.
- **BM** - BoostMap with proposed modifications, presented in section 3.5.3. The values of $|F_1|$, $|F_2|$, $|F_3|$ and $|F_4|$ were all set to N.
- **GA** - proposed Genetic embedding Algorithm using euclidean distance function, presented in section 3.6. All triples of objects were included in the training set. Parameter values were chosen as follows: $\sigma = 6$, $\alpha = 0.1$, $\beta = 0.89$, $\varphi = 1$ and $\theta = 0$. The number of dimensions, population size and the maximum number of evolutionary steps were set to 19, 1000 and 100, respectively. This enabled the algorithm to reach convergence.
- **CCA** - proposed Coevolutionary Cooperative genetic embedding Algorithm, presented in section 3.6. Parameter values were chosen as follows: $\sigma_1 = \sigma_2 = 6$, $\mu = 10$, $\alpha = 0.1$, $\beta = 0.88$, $\varphi = 1$, $\theta = 0$ and $K = 4$. The size of both populations was set to 500. The number of dimensions and the maximum number of evolutionary steps were 19 and 100, respectively.

The following sections present the homology search results for the six selected matrices. Average ROC$_{50}$ scores are specified for all the matrices. The best results are selected and plotted as differences in ROC$_{50}$ score from the original matrix and as sorted ROC$_{50}$ scores. The full search results (i.e. all ROC$_{50}$ scores for all query proteins for all matrices) are available on the enclosed CD.

4.2.1 Matrices derived from PAM70

Table 4.1: Homology search performance for matrices derived from PAM70

	PAM70	TF	NMDS	FM	BM	GA	CCA
Open penalty	18	18	18	18	18	18	18
Extend penalty	2	2	2	2	2	2	2
Average ROC_{50}	0.50	0.46	0.45	0.48	0.48	0.50	0.50
Std.dev. ROC_{50}	0.35	0.37	0.36	0.37	0.36	0.36	0.35

The homology search results obtained using matrices derived from PAM70 are shown in table 4.1. The gap open penalty was set to 18 and the extend penalty to 2 in all cases. As can be seen, both of the genetic algorithms produce matrices which give search results with the same average ROC_{50} score as the original matrix. FastMap and BoostMap also produce very good results compared to PAM70.

The plot in figure 4.2 is calculated by subtracting the ROC_{50} score of a derived matrix from the ROC_{50} score of the original matrix. As can be seen, both the TF-PAM70 and CCA-PAM70 matrices achieve the same ROC_{50} score as PAM70 on roughly half of the queries, with CCA-PAM70 being slightly more symmetric around zero. As show by the plot of sorted ROC_{50} values in figure 4.4 (page 73), CCA-PAM70 corresponds closely to PAM70.

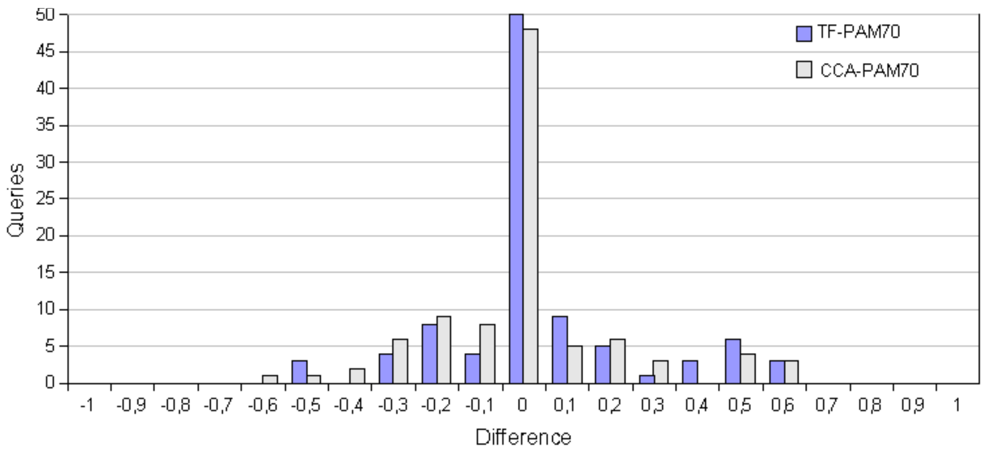


Figure 4.2: Range of difference from PAM70 ROC_{50} score

4.2.2 Matrices derived from PAM120

Table 4.2: Homology search performance for matrices derived from PAM120

	PAM120	TF	NMDS	FM	BM	GA	CCA
Open penalty	12	12	12	12	12	12	12
Extend penalty	1	1	1	1	1	1	1
Average ROC_{50}	0.59	0.44	0.42	0.49	0.47	0.49	0.51
Std.dev. ROC_{50}	0.35	0.37	0.36	0.36	0.36	0.35	0.36

As table 4.2 shows, the results for matrices derived from PAM120 show less degree of information preservation than the ones derived from PAM70. The co-operative coevolutionary algorithm is again able to produce the most sensitive metric distance matrix, closely followed by the single-population genetic algorithm and FastMap. Non-metric multidimensional scaling again produces the least sensitive metric matrix.

In this case, the information loss is easily observable from the difference plot in figure 4.3. Both TF-PAM120 and CCA-PAM120 are strongly biased towards the right side, which indicates that the original matrix scored better. However, the CCA-PAM120 matrix seems to follow a narrower curve than TF-PAM120. This is confirmed by the difference in average ROC_{50} score. Figure 4.5 on page 73 clearly shows that some of the sensitivity of the original matrix is lost, since the area under the plot of CCA-PAM120 is smaller than the area under the plot of PAM120.

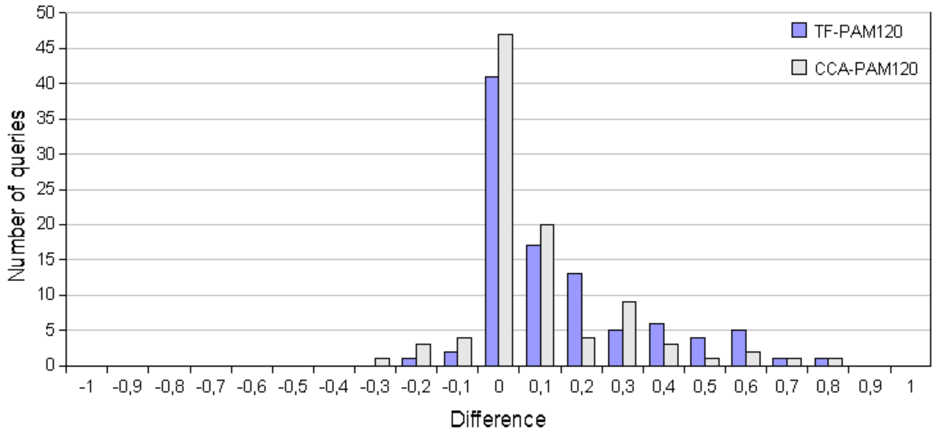


Figure 4.3: Range of difference from PAM120 ROC_{50} score

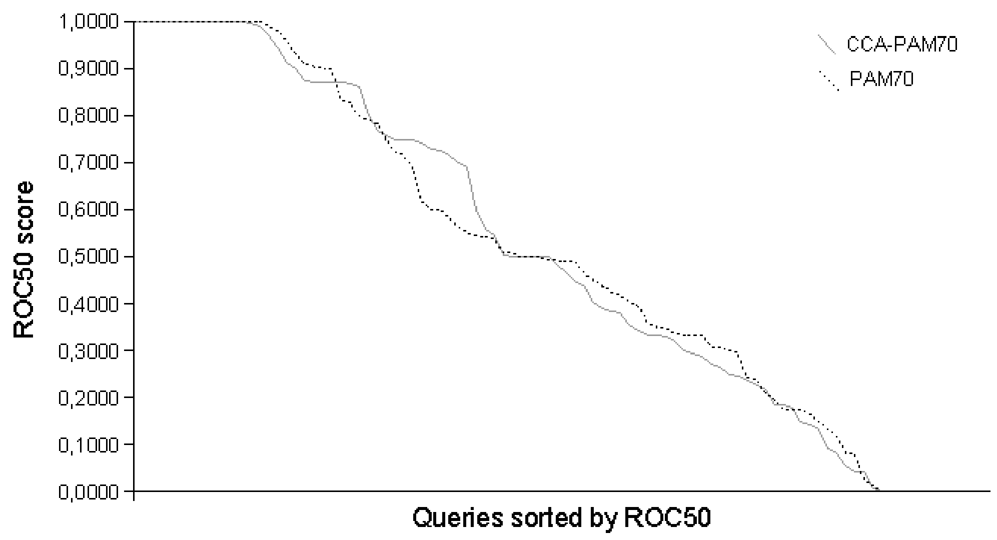


Figure 4.4: Plot of sorted ROC$_{50}$ scores for PAM70 and CCA-PAM70

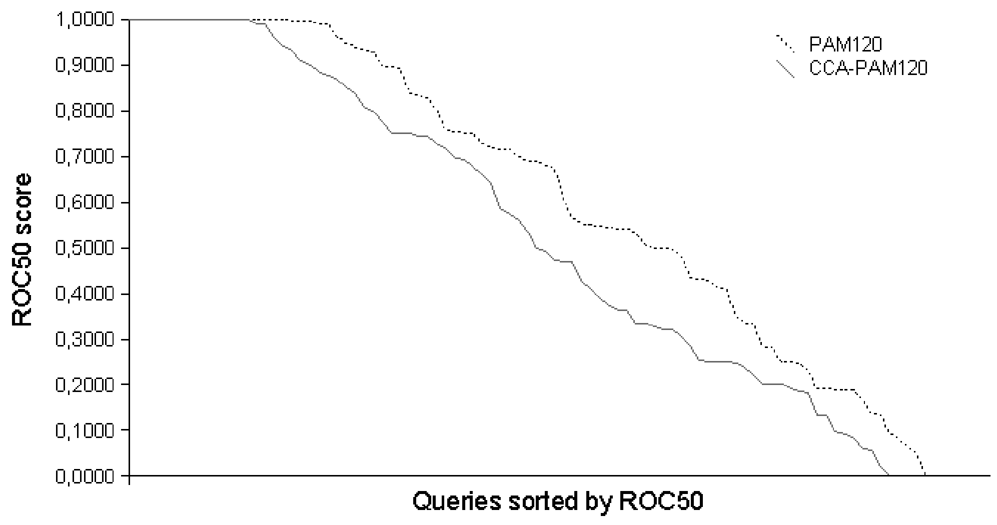


Figure 4.5: Plot of sorted ROC$_{50}$ scores for PAM120 and CCA-PAM120

4.2.3 Matrices derived from PAM250

Table 4.3: Homology search performance for matrices derived from PAM250

	PAM250	**TF**	**NMDS**	**FM**
Open penalty	12	10	10	10
Extend penalty	2	1	1	1
Average ROC_{50}	0.62	0.38	0.47	0.44
Std.dev. ROC_{50}	0.35	0.36	0.36	0.36
	mPAM250	**BM**	**GA**	**CCA**
Open penalty	10	9	10	10
Extend penalty	1	1	1	2
Average ROC_{50}	0.48	0.49	0.53	0.49
Std.dev. ROC_{50}	0.36	0.37	0.37	0.36

Table 4.3 gives the results produced by matrices derived from PAM250. In this case, the metric mPAM250 matrix mentioned in section 2.6 is also included. As can be seen, one of the evolutionary algorithms (GA) is again capable of producing the most sensitive metric matrix. Interestingly, both GA-PAM250, CCA-PAM250 and the metric matrix produced by the modified BoostMap algorithm perform better than the mPAM250 matrix. In the case of GA-PAM250, both the difference plot in figure 4.6 and the plot of sorted ROC_{50} scores in figure 4.8 on page 76 confirms that it has a sensitivity somewhere in between PAM250 and mPAM250.

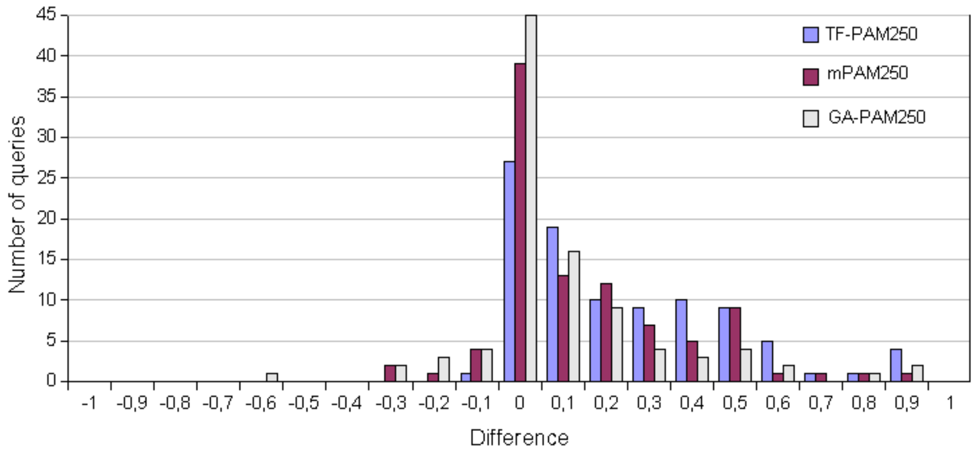


Figure 4.6: Range of difference from PAM250 ROC_{50} scores

4.2.4 Matrices derived from BLOSUM80

Table 4.4: Homology search performance for matrices derived from BLOSUM80

	BL80	**TF**	**NMDS**	**FM**	**BM**	**GA**	**CCA**
Open penalty	12	12	12	12	12	12	12
Extend penalty	2	2	2	2	2	2	3
Average ROC_{50}	0.64	0.54	0.49	0.52	0.54	0.52	0.53
Std.dev. ROC_{50}	0.33	0.35	0.36	0.35	0.34	0.34	0.35

As can be seen from table 4.4, the homology search results obtained when searching with matrices derived from BLOSUM80 are relatively similar. All of the derived matrices, with the exception of the one produced by non-metric multidimensional scaling, score roughly 0.1 less than the original BLOSUM80 matrix. In contrast to the PAM-derived matrices, the triangle fixing algorithm is able to generate a matrix which achieves a performance equal to or better than the performance of the embedding algorithms.

The difference plot in figure 4.7 clearly indicates that TF-BLOSUM80, CCA-BLOSUM80 and BM-BLOSUM80 are almost equal in performance. The bias towards the right side of zero, however, shows that some sensitivity from BLOSUM80 is lost. This is confirmed by the plot of sorted ROC_{50} scores in figure 4.9 (page 76), which also shows that TF-BLOSUM80 and BM-BLOSUM80 have almost identical score profiles.

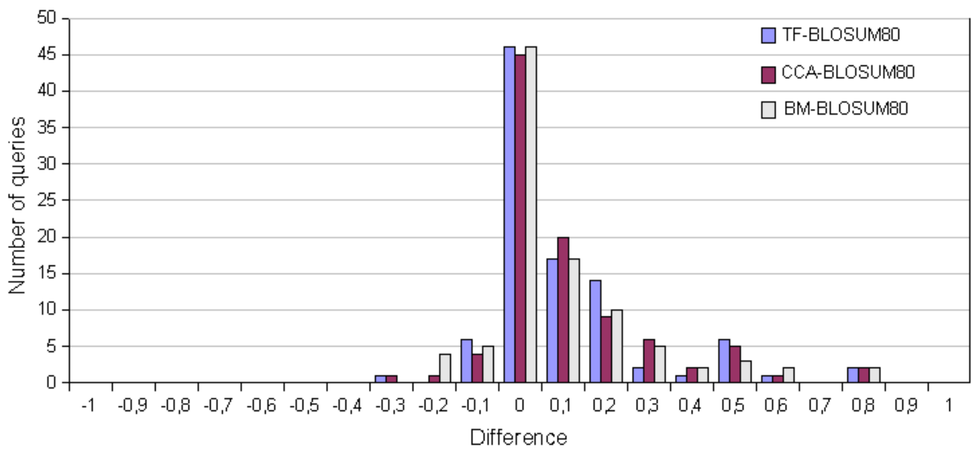


Figure 4.7: Range of difference from BLOSUM80 ROC_{50} scores

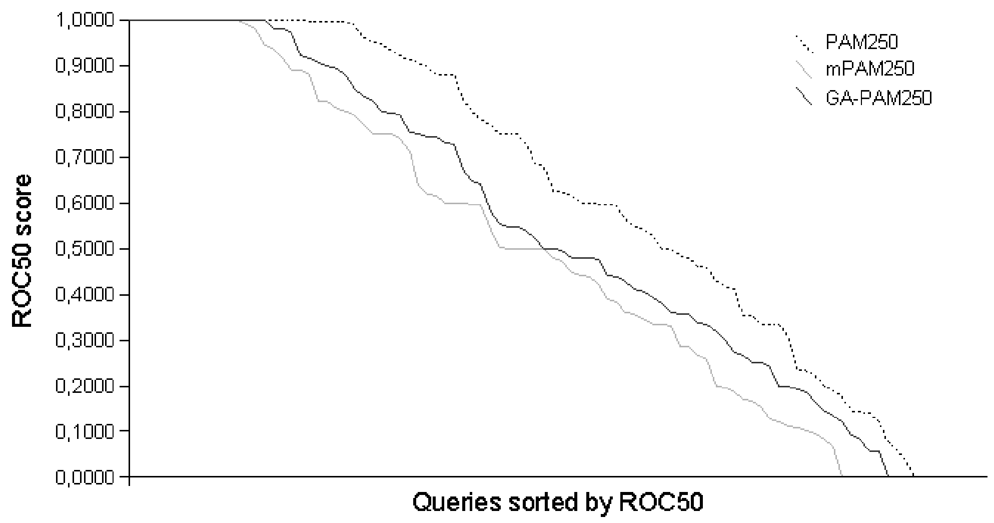


Figure 4.8: Plot of sorted ROC$_{50}$ scores for PAM250, mPAM250 and GA-PAM250

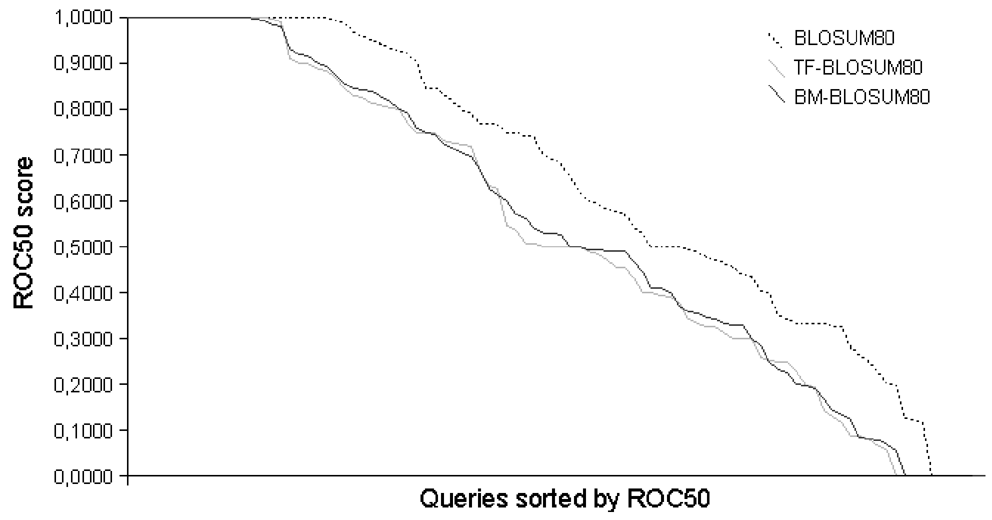


Figure 4.9: Plot of sorted ROC$_{50}$ scores for BLOSUM80, TF-BLOSUM80 and BM-BLOSUM80

4.2.5 Matrices derived from BLOSUM62

Table 4.5: Homology search performance for matrices derived from BLOSUM62

	BL62	TF	NMDS	FM	BM	GA	CCA
Open penalty	8	8	8	8	8	8	8
Extend penalty	1	1	1	1	1	1	1
Average ROC_{50}	0.66	0.52	0.50	0.52	0.54	0.49	0.50
Std.dev. ROC_{50}	0.32	0.35	0.35	0.35	0.35	0.35	0.34

The search performance of matrices derived from BLOSUM62, shown in table 4.5, shows a large degree of resemblance to the BLOSUM80 results. Again, the algorithms are almost equal in performance. The matrix produced by the triangle fixing algorithm is only beaten by BM-BLOSUM62 in terms of average ROC_{50} score.

Figure 4.10 clearly illustrates that a significant fraction of sensitivity from the original BLOSUM62 matrix is lost in the conversion process. TF-BLOSUM62 and BM-BLOSUM62 score better than BLOSUM62 in only on a very small number of queries, the original matrix scores better than the derived matrices on the majority of the queries. Figure 4.13 shows this from another point of view; the area under the BLOSUM62 curve is significantly larger than the areas under the two other curves.

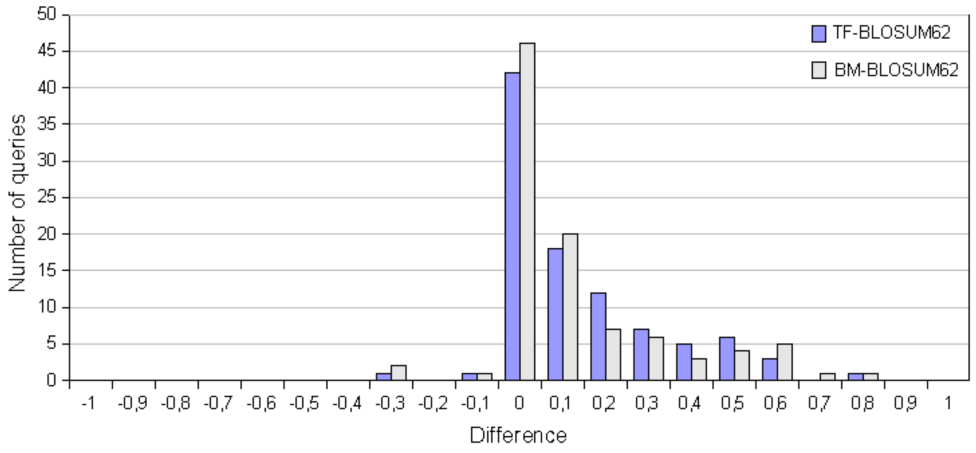


Figure 4.10: Range of difference from BLOSUM62 ROC_{50} scores

4.2.6 Matrices derived from BLOSUM40

Table 4.6: Homology search performance for matrices derived from BLOSUM40

	BL40	TF	NMDS	FM	BM	GA	CCA
Open penalty	10	10	10	10	10	10	10
Extend penalty	1	1	1	1	1	1	1
Average ROC_{50}	0.62	0.46	0.49	0.48	0.51	0.51	0.50
Std.dev. ROC_{50}	0.32	0.36	0.36	0.37	0.34	0.35	0.34

Results obtained with matrices derived from BLOSUM40 are shown in table 4.6. As can be seen, the three algorithms which are based on minimizing the number of triangular misclassification (i.e. BoostMap and the two embedding algorithms) perform marginally better than the other approaches.

The difference plot in figure 4.11 shows that both GA-BLOSUM40 and BM-BLOSUM40 perform significantly better than the matrix derived using the triangle fixing algorithm, since they score equal to BLOSUM40 on a higher number of queries and seem to follow a narrower curve. Figure 4.12 shows that although there are some difference in the curve profiles between GA-BLOSUM40 and BM-BLOSUM40, they perform roughly equal because the areas under the curves are approximately equal.

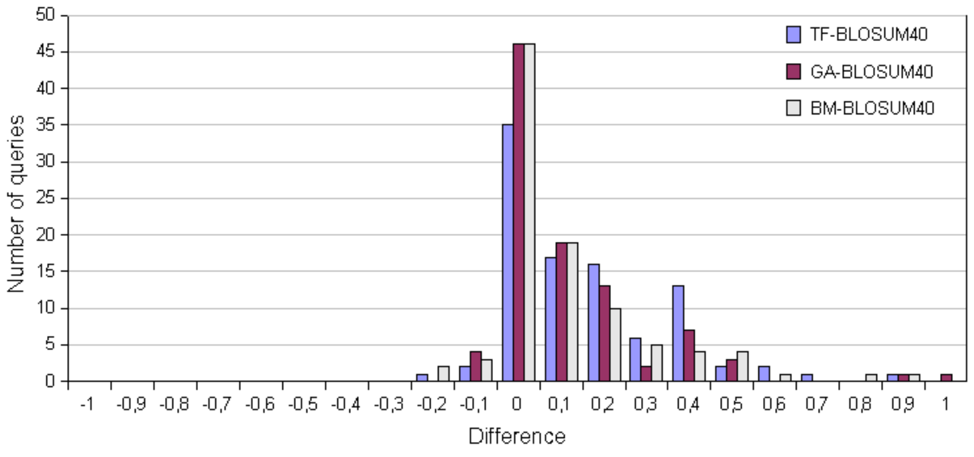


Figure 4.11: Range of difference from BLOSUM40 ROC_{50} scores

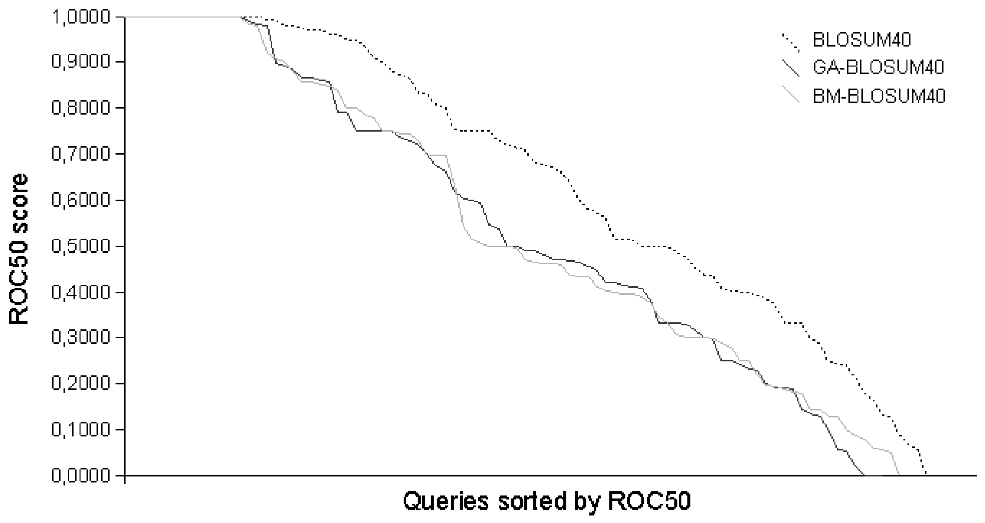


Figure 4.12: Plot of sorted ROC_{50} scores for BLOSUM40, GA-BLOSUM40 and BM-BLOSUM40

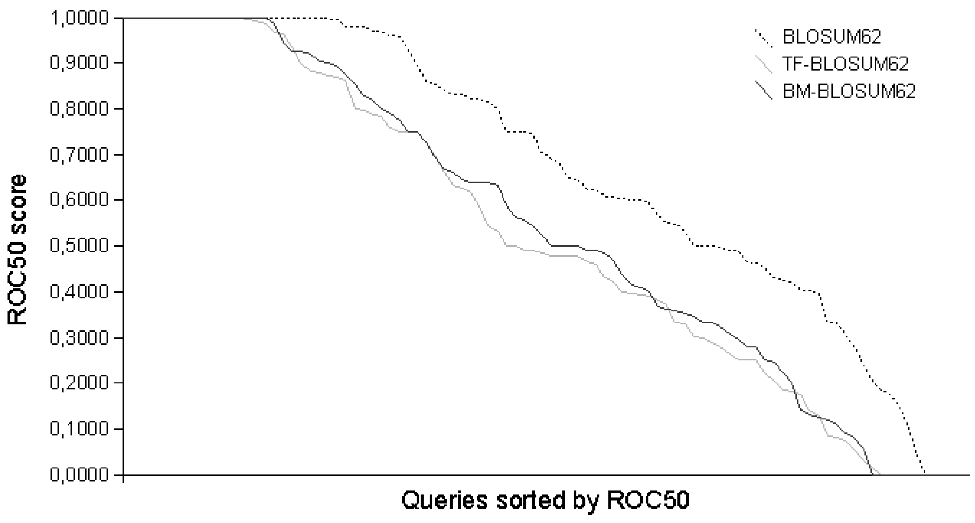


Figure 4.13: Plot of sorted ROC_{50} scores for BLOSUM62, TF-BLOSUM62 and BM-BLOSUM62

4.3 Test for statistical significance

The results presented in the previous sections can be used to test whether the metric matrices produced by the proposed methods perform as well as the original matrices, from a statistical point of view. Assuming that both the database to be searched in and the substitution matrix are fixed, the distribution of ROC_{50} scores depends on properties of the query proteins. This distribution is not known in advance, so we cannot apply tests for statistical significance based on defined probability distributions. Thus, a non-parametric test must be used.

When comparing the ROC scores obtained using a metric matrix with the ROC scores obtained using the original matrix, we are essentially dealing with paired observations. This means that the *sign test* presented in [WMMY02] can be applied. A sign test is used to test hypotheses on population *medians* rather than means. If M_{original} and M_{metric} are used to denote the median ROC score obtained using the original and the metric matrix, respectively, the following hypothesis test can be formulated:

$$\begin{aligned} H_0 &: M_{\text{metric}} = M_{\text{original}} \\ H_1 &: M_{\text{metric}} < M_{\text{original}} \end{aligned}$$

To test the hypothesis, we start by subtracting M_{original} from each ROC score obtained with the metric matrix, and then take the sign of the resulting values. If H_0 holds, we expect the number of plus and minus signs to be approximately equal. Since we are now dealing with a distribution of binary values, this can be tested using a binomial distribution where the success probability of a trial is 0.5 under H_0. Representing the number of plus signs by x, the probability of obtaining x or less can be calculated as

$$P = P(X \leq x \text{ when } p = 0.5) = \sum_{i=0}^{x} b(i; 96, 0.5)$$

where 96 is the number of queries. This P value can then be used to either accept or reject H_0 at some level of significance. Table 4.7 shows the P values of homology search results in this work.

Table 4.7: P values of search results

	TF	**NMDS**	**FM**	**BM**	**GA**	**CCA**
PAM70	0.238	0.131	0.459	0.380	0.459	0.459
PAM120	0.0037	0.0028	0.063	0.026	0.0052	0.075
PAM250	$1.1 \cdot 10^{-6}$	0.0028	$6.6 \cdot 10^{-6}$	0.016	0.026	0.026
BLOSUM80	0.041	0.0052	0.016	0.041	0.016	0.016
BLOSUM40	0.0028	0.0092	0.0092	0.016	0.0092	0.016
BLOSUM62	0.0092	0.0014	0.0092	0.0092	0.0007	0.0028

Chapter 5

Analysis and discussion

This chapter discusses the experimental results presented in chapter 4. An analytical comparison of the proposed methods is also given. We are mainly interested in finding patterns in the empirical results which can be related to the design of the algorithms. This will enable questions such as the ones posed at the end of chapter 3 to be answered.

The first section of the chapter gives a formal analysis and comparison of the methods and algorithms in terms of design and computational complexity. Then the empirical results are discussed. Finally, the discussions are synthesized and summarized in the last section.

5.1 Analytical comparison of methods

Differences between the proposed methods can be discussed both in terms of computational resource usage and in terms of general advantages and disadvantages. As for comparing algorithms in terms of computational resource usage, first note that the complexity of the algorithms has no connection to the complexity of homology search methods which *use* the produced metric matrix. Once such a matrix has been generated, it can be used to index the data using a method like the ones presented in section 2.5. The algorithm for producing the metric matrix is only run once, and search procedures subsequently utilize the metricity of the output matrix directly.

However, since the methods are intended to be generic to the problem of making a matrix metric, one may come across problems where the complexity of the algorithms is crucial:

- If the algorithm needs to be run repeatedly, perhaps as part of a looping procedure, its computational complexity is obviously crucial.

- If the size of the input matrix is large, algorithms of high computational complexity may not be able to produce results within acceptable time limits. In this work 20×20 amino acid substitution matrices have been considered, but it is not hard to imagine that there may exist cases where one would like to apply the algorithms to larger matrices.

Therefore, the following subsection gives a brief analysis of each algorithm in terms of complexity. The algorithms are then discussed in general terms, taking the complexity into account.

5.1.1 Analysis

Table 5.1: Computational complexity of algorithms, where d is the number of dimensions, M is the number of classifiers considered at each iteration, P is the population size, K is the number of collaborators selected from the other population and c is the number of iterations needed for convergence.

Algorithm	**Complexity**
Non-metric multidimensional scaling	$c \times O(N^2 \log N^2)$
FastMap	$O(d^2N)$
BoostMap	$c \times O(MN^3)$
Genetic embedding algorithm	$c \times O(P \max(dN^2, N^3))$
Coevolutionary embedding algorithm	$c \times O(PK \max(dN^2, N^3))$
Triangle fixing algorithm	$c \times O(N^3)$

The algorithm for non-metric multidimensional scaling presented in section 3.3 iterates until convergence. In most cases, it is not possible to find an analytical expression for the total computational complexity of such algorithms. Instead, we focus on the complexity of the procedures which are run at each iteration. As can be seen in algorithm 3, the complexity of each iteration is upper bounded by the need to sort the distance values between embedded points. For an $N \times N$ matrix, there are N^2 such values. Sorting these can be done in $O(N^2 \log N^2)$ time. The total complexity of the algorithm is therefore $c \times O(N^2 \log N^2)$, where c is the number of iterations needed for convergence. This notational convention of representing the number of iterations necessary for convergence by c will be used in all following complexity formulas.

The FastMap algorithm introduced in section 3.4 is dominated by the d recursive calls to produce a d-dimensional embedding. Algorithm 4 shows that the most complex operation in each recursive call is quadratic in N. However, as explained, the steps which calculate a new distance matrix are only included for clarity in the pseudocode. Instead of computing a distance matrix in $O(N^2)$

time, the distance between two elements in an iteration can be computed "on the fly" in $O(d)$ time using equations 3.1 and 3.2. As can be seen (line 10 in algorithm 4), three such distance values are needed to compute the coordinates of N elements in a particular iteration. Thus, the total complexity of FastMap is $O(d^2N)$. Note, however, that the construction of a new distance matrix from the final embedding is $O(N^2)$.

As noted by Athitsos et al. in [AASK04b], the complexity of BoostMap is highly sensitive to the size of the training set. Recall from section 3.5 that the training set consists of triples (i, j, k) where $0 < i < j < k \leq N$. Thus, the maximum size is $T = \Theta(N^3)$ (but nevertheless adjustable to the user). If M is the number of classifiers considered for inclusion in each iteration, a total of $\Theta(MT)$ evaluations between embeddings and training triples are needed, because one evaluation of the Z_j function (equation 3.5) takes $\Theta(T)$ time. These evaluation dominate the complexity of each iteration, so the total complexity of BoostMap is $c \times O(MT)$. If all triples are included in the training set, as done in the implementation used in this work, the complexity is $c \times O(MN^3)$. Note that this analysis assumes that Z_j can be minimized with respect to α in a constant amount of time.

Each iteration of the genetic embedding algorithm proposed in section 3.6 is dominated by the computation of embedding costs, assuming that the tournament selection procedure picks a relatively small constant number of embeddings (7 was used in this work). Algorithm 11 shows that computing the cost of an embedding is $\Theta(\max(dN^2, N^3))$, where d is the number of dimensions. The computation of N^2 euclidean distances is $\Theta(dN^2)$, while the computation of triangular misclassifications is $\Theta(N^3)$ since all triples (i, j, k) of elements where $0 < i < j < k \leq N$ are considered. If P is the desired population size, the complexity of the algorithm is $c \times O(P \max(dN^2, N^3))$.

The cooperative coevolutionary embedding algorithm presented in section 3.6 is the most complex one in terms of resource usage, since it attempts finding a near-optimal embedding while at the same time finding a near-optimal metric distance function. This algorithm is also dominated by the computation of embedding costs. As can be seen in algorithm 14, a Mahalanobis matrix or an N-dimensional embedding is compared with K individuals from the other population. Computing the distance between two points takes $\Theta(d^2N^2)$ time if implemented straightforward. However, since it is known in advance that Z is a matrix with non-negative values on the diagonal and zeros elsewhere, the computation can be achieved in $\Theta(dN^2)$ time. As with the single-population genetic algorithm, computing triangular misclassifications is $\Theta(N^3)$. Thus, the complexity of computing the cost of one matrix or embedding is $\Theta(K \max(dN^2, N^3))$. Assuming that a small constant number of elements are picked at random by the tournament selection procedure, this cost computation step is dominant in each iteration. The full complexity of the algorithm is $c \times O(PK \max(dN^2, N^3))$.

Algorithm 2 shows that each iteration of the triangle fixing algorithm apply correction terms to each triangle (a, b, c). For an $N \times N$ matrix, there are $O(N^3)$

such triangles. Thus, the total complexity of the algorithm is $c \times O(N^3)$.

Table 5.1 summarizes the computational complexities of the algorithms.

5.1.2 Comparison and discussion

As the analysis in the previous section shows, FastMap is the best choice when only considering computational complexity. It is linear in N, while the other algorithms are quadratic or cubic. This means that it will be able to handle much larger matrices. FastMap will outperform the other algorithms even when taking into account the N^2 steps necessary to construct the final metric substitution matrix from the d-dimensional embedding. All the other algorithms require some steps to be run repeatedly until some termination condition is met. It is difficult to compare the complexity of these directly, since no analytical expressions for the termination conditions are available. It is, however, clear that the evolutionary algorithms require more computational resources than the other ones in most cases. The population size is typically selected to be much larger than N, leading to computation times in order of minutes even for small N.

Other factors must of course be taken into account when comparing the algorithms. The performance of the metric matrix produced by the algorithm, to be discussed in section 5.2, is obviously essential. But there are a number of inherent differences between the algorithms which are worth pointing out:

- As mentioned in chapter 3, the triangle fixing algorithm is conceptually different from the others in that it constructs a metric matrix by *explicitly* enforcing triangle constraints. Metricity is *implicit* in the other algorithms.
- If the substitution matrix to be converted is a similarity matrix (as in this work), both the triangle fixing algorithm and FastMap require this matrix to be transformed into a dissimilarity matrix first. Section 3.5.3 showed that it is questionable how much information such a transformation is able to preserve. The problem is that diagonal cell values of matrices like amino acid substitution matrices may not be equal, so the transformation is not as easy as subtracting the maximum matrix value from each cell value. The other four algorithms takes this into account by utilizing the data in the similarity matrix directly. In the case of non-metric multidimensional scaling and BoostMap, some modifications to the original matrices have been proposed in this work for the purpose of handling similarity matrices.
- Non-metric multidimensional scaling, FastMap and the first genetic embedding algorithm require a metric distance function to be chosen and fixed before the metric distance matrix is recovered. The cooperative coevolutionary embedding algorithm introduces a new degree of freedom by attempting to evolve a population of distance functions in parallel to the

population of spatial embeddings. BoostMap also uses a distance function which is determined dynamically by the algorithm; recall that the distances recovered from the final embedding is computed using a weighted Manhattan distance, where the weights of the axis are learned by the algorithm. An explicit choice of distance function is not relevant to the triangle fixing algorithm.

- BoostMap determines the dimensionality of the metric space dynamically. Recall that each one-dimensional embedding selected by the algorithm represents one dimension of the final embedding, and that the algorithm has the ability to both add and remove such one-dimensional embeddings during execution. All the other embedding algorithms require a fixed number of dimensions to be specified in advance. Again, the problem is not relevant to the triangle fixing algorithm.
- Non-metric multidimensional scaling, the two genetic/evolutionary algorithms and BoostMap all construct a spatial embedding by minimizing some cost function. The first three of these four can converge into a local minima of the cost function, and the author is not aware of any proof which shows that this is not also the case for BoostMap. This is clearly a disadvantage. Especially the cooperative coevolutionary embedding algorithm was found to be affected by this problem, requiring a lot of parameter tuning and trials before being able to produce better embeddings than the single-population genetic algorithm. FastMap does not minimize any cost function, so we cannot speak in concrete terms about local minima for this algorithm. As for the triangle fixing algorithm, it can be shown that it computes the globally optimal solution to the *metric nearness* problem defined in section 3.1 [DST04]. It should be obvious that being able to avoid local minima is highly desirable, especially in cases where successive trials is not an option.

The differences between the algorithms discussed above is expected to have an effect on the ability to preserve the information in the original substitution matrices. Possible effects are uncovered in sections 5.2 and 5.3, where the algorithms are discussed in light of homology search results.

Some justifications should be made regarding the actual choice of algorithms, and possible alternative choices should be pointed out:

- Among the algorithms used, all but the two evolutionary ones are based on existing works. These represent only a subset of the many embedding algorithms available. An overview of some alternative algorithms can be found in [HS03]. Multidimensional scaling was selected because of its widespread usage. This was also the case for FastMap, which seems to have gained much popularity since its introduction in 1995. BoostMap was selected because of its idea of minimizing the number of triangular misclassifications, which in turn was the inspiration behind the two evolutionary algorithms and the modifications of the original BoostMap algorithm. As mentioned in the introduction to chapter 3, the triangle fixing algorithm was chosen so that the idea of deriving distance metrics from spatial embeddings could be compared with a conceptually different approach.

- Both FastMap and the single-population genetic algorithm allows an arbitrary metric distance function to be used for recovering a distance matrix. In this work, only the classical euclidean distance function has been considered. The main motivation for this has been results from preliminary tests showing that the methods were outperformed by other algorithms regardless of distance function. In the case of the single-population genetic algorithm, experimentation with alternative distance functions ultimately resulted in the design of the coevolutionary algorithm. This coevolutionary approach was then expected to outperform the single-population approach in any case.

- The class of Mahalanobis matrices used to represent metric distance functions in the coevolutionary algorithm is restricted to matrices with non-negative values on the diagonal and zeros elsewhere. In theory, any positive semi-definite matrix can be used. Thus, the limitation could be removed if an efficient crossover operator was found which guarantees that the offspring matrix is positive semi-definite.

5.2 Homology search results

Chapter 4 presented homology search results for six commonly used amino acid substitution matrices. As explained in section 4.1.4, these were selected to represent a wide range of divergence between protein sequences. Based on these empirical results, we are mainly interested in knowing:

- Are the metric matrices produced by the proposed methods able to retain the sensitivity of the original matrices?

- Which algorithms produce the best matrices in terms of sensitivity preservation? How do the algorithms based on deriving distance metrics from spatial embeddings compare with the "straightforward" triangle fixing algorithm?

- Do algorithms perform equally well on all matrices? If not, are there any apparent patterns in terms of matrix family (PAM or BLOSUM) or target divergence?

This subsection summarizes and discusses the homology search results in light of these questions. Results from the six chosen substitution matrices are discussed in separate sections. A summary of the findings is given in section 5.3.

5.2.1 PAM matrices

PAM70

All of the algorithms seem to work well on the PAM70 matrix. In fact, matrices produced by the two proposed genetic embedding algorithms are able to produce search results with the same average ROC_{50} score as the original matrix. They are closely followed by FastMap and BoostMap, which produce results with a marginally lower ROC_{50} score. With the exception of non-metric multidimensional scaling, the triangle fixing algorithm performs slightly worse than the embedding algorithms. Nevertheless, the P values presented in table 4.7 show that the level of significance must be chosen larger than 0.131 to be able to reject the null hypothesis (see section 4.3) for any of the matrices. Recall that the P value tests the hypothesis by assuming that the observed ROC_{50} scores for the original and the derived matrices are samples from the same distribution. Using a significance level of 0.05 or 0.10, this cannot be rejected for any of the matrices.

In the case of the CCA-PAM70 matrix (i.e. the metric matrix produced by the cooperative coevolutionary algorithm), the difference plot in figure 4.2 confirms this. About half of the score values are equal to the ones obtained using the original PAM70 matrix. Among the ones which are not, about half of them are lower and half of them are higher than the PAM70 scores. The plot of sorted ROC_{50} scores in figure 4.4 also shows that the two matrices are close to equal in performance.

This apparent ability of the metric matrices to retain the sensitivity of the original matrix was actually quite unexpected. Previous attempts to convert amino acid substitution matrices into metric distance matrices have proved that some property loss is inevitable. Recall that the two evolutionary algorithms work by trying to minimize the number of triangular misclassifications in relation to the original substitution matrix. BoostMap takes a similar approach, by using triangular misclassifications to evaluate and select weak classifiers. Calculations of the triangular misclassifications made by the final metric matrices show that CCA-PAM70 makes 109 misclassifications, GA-PAM70 makes 111 misclassifications and BM-PAM70 makes 171 misclassifications. There are a total of 1140 triples (i, j, k) where $0 < i < j < k \leq 20$, so the numbers represent a small but significant fraction of all possible misclassifications.

This suggests, as might be expected, that there is no *direct* equivalence between preservation of sensitivity and preservation of triangular classifications. The average ROC_{50} score of PAM70 is only 0.50, which is a relatively modest performance. It means that a relatively large amount of false positives are ranked ahead of the true positives in many query results. An explanation to the sensitivity preservation may therefore be that the produced metric matrices are able to correct some of the mistakes made by the original PAM70 matrix. Whether this has a formal explanation in the design of the algorithms or has happened by chance in the case of PAM70 has not been further investigated. In any case, the results show that the two evolutionary algorithms, BoostMap and FastMap are able to produce metric matrices which preserves the sensitivity of PAM70.

Because of its good performance, the entire CCA-PAM70 matrix has been included in appendix B. In the figure, the matrix produced by the cooperative coevolutionary algorithm has been scaled by a factor of 0.05 and rounded to one decimal.

PAM120

Contrary to the case of PAM70, none of the algorithms are able to produce a metric matrix with the same sensitivity as the PAM120 substitution matrix. Closest in average ROC_{50} score is the matrix produced by the cooperative coevolutionary algorithm, with a score of 0.51. This is 0.08 lower than PAM120. Nevertheless, this suggests that a large fraction of the sensitivity is preserved. In relation to the results obtained with the PAM70 matrix, two key observations can be made:

- The rank order of the matrices produced by the different algorithms (in terms of average ROC_{50} score) seems to be similar. In both cases, all embedding algorithms except for non-metric multidimensional scaling outperforms the triangle fixing algorithm. Furthermore, the cooperative coevolutionary algorithm produces the best matrices, while FastMap, BoostMap and the single-population genetic algorithm are roughly equal in performance.

- Triangular misclassifications made by the PAM120-derived matrices suggest that there may be some connection to ROC_{50} performance. CCA-PAM120 makes 141 misclassifications, GA-PAM120 makes 132 misclassifications and BM-PAM120 makes 180 misclassifications. In comparison, NMDS-PAM120 makes 302 misclassifications. However, triangular misclassifications do not dictate ROC_{50} performance alone. The fact that FM-PAM120 makes 260 misclassification while still performing equal to GA-PAM120 is an indication of this.

Table 4.7 shows that we can reject the null hypothesis and conclude that none of the derived matrices have sensitivities equal to PAM120 if a significance level of 0.05 is chosen. If we choose 0.10 as significance level, the null hypothesis cannot be rejected in the cases of CCA-PAM120 and BM-PAM120. Figure 4.3 confirms that the best performing matrix, CCA-PAM120, preserves much of the sensitivity from PAM120, although the difference plot seems to be slightly biased towards positive values (i.e. PAM120 ROC_{50} score higher than CCA-PAM120 ROC_{50} score). The plot of sorted scores in figure 4.5 confirms this. Surprisingly, GA-PAM120 is still rejected even though its average ROC_{50} score is equal to the score of the matrix produced by BoostMap. This is because the hypothesis test uses the median instead of average ROC_{50}, and shows that GA-PAM120 has a distribution of values which is more biased towards one side of the PAM120 median value than CCA-PAM120 and BM-PAM120.

PAM250

In terms of sensitivity preservation, the PAM250-derived matrices seem to resemble the ones derived from PAM120. Again, the three algorithms based on preserving triangular classification produce the most sensitive metric matrices. The number of misclassifications in this case are 131 for the matrix produced by the single-population genetic algorithm, 136 for the matrix produced by the cooperative coevolutionary algorithm and 162 for the matrix produced by BoostMap.

As in the cases of PAM70 and PAM120, matrices produced by the embedding algorithms outperform the matrix produced by the triangle fixing algorithm by a significant amount. The difference in average ROC_{50} score between TF-PAM250 and GA-PAM250 is 0.15. In contrast, the difference in average score between GA-PAM250 and the original matrix is only 0.09. This latter difference is still, however, not small enough to claim that all the sensitivity of PAM250 has been preserved. As table 4.7 shows, we would have to choose a level of 0.026 or lower for any of the results to be statistically significant. Since significance levels are commonly set to 0.10 or 0.05, it would be very optimistic to claim that any of the derived matrices perform equal to the original matrix.

Not considering full sensitivity preservation, the matrices produced by the two evolutionary algorithms and BoostMap nevertheless seem to be good approximations. It is interesting to note that these three matrices produce search results with higher average ROC_{50} scores than the mPAM250 matrix recently proposed by Xu and Miranker in [XM04]. To compare mPAM250 and GA-PAM250, the full ROC curves were calculated and plotted. They are shown in figure 5.1.

ROC curves show the average fraction of true positive hits returned per number of false hits. Thus, the area under the curve can be seen as a measure of sensitivity performance. As figure 5.1 shows, the GA-PAM250 matrix produced by the single-population genetic algorithm performs better than the mPAM250 matrix. All three curves have the same shape, which shows that both of the two metric matrices have characteristics similar to PAM250.

In relation to the results with PAM70 and PAM120, it is worth noticing that non-metric multidimensional scaling performs better in this case. Another difference is that the cooperative coevolutionary algorithm is not able to produce a metric

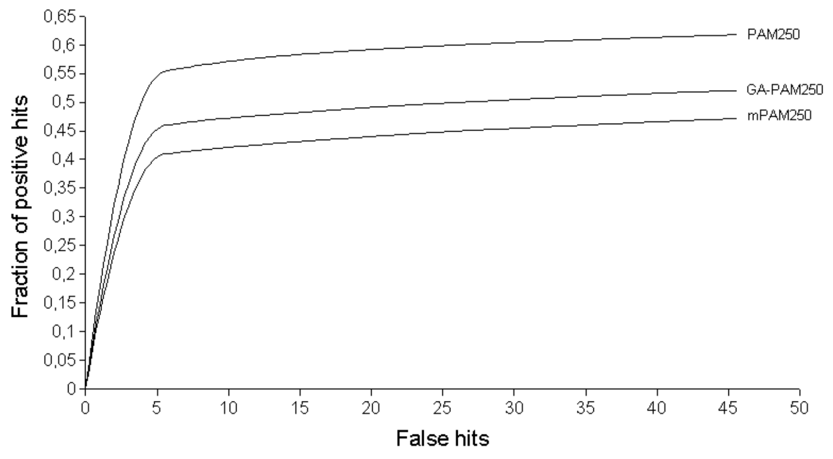


Figure 5.1: ROC curves for PAM250, mPAM250 and GA-PAM250

matrix which performs better than the one produced by the single-population genetic algorithm. Finding an explanation to these facts is hard, since a formal connection between the philosophy of the algorithms and the data in the substitution matrices is yet to be found. It simply shows that the inherent structure of the input matrix has an impact on the performance of the output matrix. Local minima are also possibly part of the explanation, since both non-metric multidimensional scaling and the cooperative coevolutionary algorithm are exposed to this problem.

Because of its good performance in relation to other attempts to construct a metric matrix based on PAM250, the GA-PAM250 is included in its entirety in appendix C.

PAM matrices - a summary

The main observations which can be extracted from the homology search results with PAM matrices are:

- In the case of PAM70, the two evolutionary algorithms are able to produce metric matrices which preserve all the sensitivity of the original matrix.
- In all three cases, the two evolutionary algorithms produce matrices performing better than matrices produced by the other algorithms.
- FastMap, BoostMap and the two evolutionary algorithms perform better than the triangle fixing algorithm in all three cases. FastMap is able to produce well-performing distance matrices even though its mathematical foundation requires the input matrix to be metric.

5.2.2 BLOSUM matrices

BLOSUM80

The results from homology search using metric matrices derived from the BLOSUM80 matrix are quite different from results with PAM-derived matrices. As table 4.4 shows, the difference in average score is marginal between different algorithms. The triangle fixing algorithm and BoostMap produce the most sensitive matrices, while the matrix derived using non-metric multidimensional scaling performs worst. The difference in average score from the original matrix varies between 0.10 and 0.15, which must be considered to be a significant difference. This is confirmed by table 4.7, which shows that none of the results have a statistical significance which is high enough to claim that the entire sensitivity of the BLOSUM80 matrix is preserved (assuming a significance level of 0.05).

Nevertheless, the matrices produced by all of the algorithms are able to preserve a large amount of sensitivity.

The possible connection between sensitivity preservation and number of triangular misclassifications seems not to apply in this case. Calculations show that BM-BLOSUM80 makes 161 misclassifications, while the corresponding numbers for GA-BLOSUM80 and CCA-BLOSUM80 are 129 and 133, respectively. Still the BoostMap-generated matrix is able to outperform matrices produced by the two evolutionary algorithms.

BLOSUM62

The results obtained with matrices derived from BLOSUM62 are very similar to the results in the case of BLOSUM80. All algorithms produce matrices of roughly equal sensitivity. BoostMap is marginally best, closely followed by the triangle fixing algorithm and FastMap. Choosing a significance level of 0.05, none of the produced matrices can be claimed to have the same performance as the original matrix.

An investigation of the triangular misclassifications made by the three algorithms which are based on minimizing this measure confirms that the connection to sensitivity preservation seems to be very loose in the case of BLOSUM matrices. The number of misclassifications is 253 for BM-BLOSUM62, 233 for GA-BLOSUM62 and 230 for CCA-BLOSUM62. These scores correspond neither with the ranking of the average ROC_{50} scores, nor with the actual value of the scores. Thus, the patterns observed with the PAM matrices seem not to be present in the case of BLOSUM matrices.

BLOSUM40

Average ROC_{50} scores from homology search using BLOSUM40-derived matrices shows a higher level of resemblance to the PAM results than BLOSUM80 and BLOSUM40. Matrices derived using spatial embedding algorithms are roughly equal in performance, while the matrix derived using the triangle fixing algorithm performs slightly worse. Thus, there seems to be an advantage in choosing one of the embedding algorithms in this case.

Nevertheless, differences in sensitivity between different matrices are not as high as with the PAM matrices. For example, observe that the difference between the most sensitive and least sensitive matrix is 0.15 in the case of PAM250 (see table 4.3). In this case, the difference is only 0.05. Even though the embedding algorithms are marginally better than the triangle fixing algorithm in this case, it confirms that the differences between the conceptually different methods are questionable in the case of BLOSUM matrices.

BLOSUM matrices - a summary

The main observations which can be extracted from the homology search results with PAM matrices are:

- The advantage of choosing an embedding algorithm over the triangle fixing algorithm is more questionable than in the case of PAM matrices.
- Differences between matrices produced by the different algorithms are smaller than with PAM matrices.
- All of the produced matrices are able to preserve a significant part of the sensitivity of the original matrix.

5.3 Synthesis and summary

As discussed in the previous section, the experimental results suggest that the effects of the produced metric matrices depend on the family of the input matrix:

- When dealing with matrices from the PAM family, there is an advantage in choosing one of the algorithms which derive a metric distance matrix from a spatial embedding over the triangle fixing algorithm. Furthermore, the two genetic algorithms proposed in this work outperform the other embedding algorithms by a slight amount.
- When dealing with matrices from the BLOSUM family, differences between the embedding algorithms and the triangle fixing algorithm are less clear.

The most likely explanation to these differences is that there are inherent difference between data contained in matrices from the two families. As shown in section 2.3, there are fundamental differences in the way they are derived from empirically collected data.

An interesting aspect of the average ROC_{50} scores is the fact that they seem to be relatively similar across the six mutation matrices. Based on this, one might wonder if the results are somehow effects of randomness. The difference plots presented in chapter 4 show that this is most probably not the case. For many of the produced matrices, the difference in ROC_{50} from the original matrix is zero for almost half of the protein queries. This, in turn, shows that there is a significant amount of correlation between the metric matrices and original ones.

The similarity in average ROC_{50} score also raises the question: Does this score level (just above 0.5) represent a maximum amount of sensitivity which can be preserved from a matrix from the PAM and BLOSUM families? This work does not attempt to answer this question.

Some key observations can also be pointed out in relation to the design of the different algorithms:

- Using matrices from the PAM family, BoostMap and the two evolutionary algorithms are among the best performers. Recall that these three algorithms are based on minimizing the number of triangular misclassifications. This suggests that triangular misclassifications are suitable measures of cost when constructing metric substitution matrices from spatial embeddings.
- The idea of using difference in rank between distance and similarity values, as non-metric multidimensional scaling does, seems not to work as well as using triangular misclassifications.
- Even though FastMap requires input matrices to satisfy the triangle inequality, it produces metric embeddings from which well-performing matrices can be derived.
- The difference in performance between the single-population genetic algorithm and the coevolutionary one is only marginal. The most likely explanation to this is the large difference in search space sizes between the algorithms. Since the coevolutionary algorithm must search in a space which is exponentially larger than that of the single-population algorithm, finding a near-optimal solution is obviously a more difficult problem.

These observations suggest that the evolutionary algorithms proposed in this work are appropriate for matrices of modest size, like amino acid substitution matrices. As the size of input matrices increases, the computational resources demanded by the algorithms will reach levels which are not feasible. In such cases, BoostMap or FastMap are better choices. Note, however, that the results from the PAM and BLOSUM families of substitution matrices show that the appropriateness of the algorithms is problem dependent. It is therefore difficult to give any advice as to which should should be chosen in the general case.

Finally, it should be emphasized that the algorithms proposed in this work are able to produce a metric distance matrix from PAM70 which preserves the sensitivity of the original matrix. Furthermore, the proposed genetic algorithm produced a metric matrix from PAM250 which performs better than the mPAM250 matrix presented recently.

Chapter 6

Conclusions and future work

The work presented in this report has investigated different ways of converting a non-metric matrix into a metric distance matrix. In particular, two main methods of achieving this goal have been considered and evaluated. The first method enforces metricity explicitly by manipulating data in the matrix directly. The other one is based on the general idea to embed the data in the substitution matrix into a metric space, from which a metric distance matrix is thereafter recovered. Metricity is ensured implicitly in this latter approach. Two evolutionary algorithms have been proposed in the work for performing the actual mapping of data into metric space. The algorithms are based on comparing proximities within triples of embedded elements with corresponding proximities in the non-metric matrix. They have been evaluated together with three existing embedding algorithms, two of which have been modified in this work for handling substitution matrices in the form of similarity matrices.

The algorithms and methods presented and compared are *general* in the sense that they can be applied to any substitution matrix, without regards to how the data in the matrix has been derived.

The methods have been tested on amino acid substitution matrices. These empirical tests show that the embedding approach produces the best performing metric matrices when dealing with input matrices from the PAM family. Furthermore, the proposed evolutionary algorithms outperform the other embedding algorithms by a slight amount. Very promising search results were obtained with the CCA-PAM70 and GA-PAM250 matrices. The latter one is able to outperform the mPAM250 matrix recently presented in a similar work. Results obtained from matrices derived from the BLOSUM family are less clear; there seems to be no immediate advantage in choosing an embedding algorithm instead of the triangle fixing algorithm.

This enables us to conclude that the suitability of the suggested methods is dependent on the inherent nature of the data in the original matrix. For the specific case of PAM substitution matrices, it seems to be suitable to apply the strategy of deriving a distance matrix from a metric space where an attempt is made to preserve triangular proximities. As the derived distance matrix represents a metric distance measure between proteins, it can be utilized to store protein sequences in efficient index structures. Although the entire sensitivity of the original matrix is not preserved, this could prove to be useful in cases where approximate search results are acceptable.

Future work should focus on investigating alternative ways of measuring the quality of spatial embeddings. It is quite possible that there may exist other schemes which are more effective than minimization of triangular misclassifications. Furthermore, the evolutionary embedding scheme proposed in this work should be investigated further; better ways of representing the embedding, recombining embeddings and evaluating embeddings could exist.

Appendix A

Derivation of PAM and BLOSUM matrices

This appendix explains how the PAM and BLOSUM families of matrices are derived from empirically collected data. A brief comparison of the two models is also given. Finally, a brief survey of alternative amino acid substitution matrix models rounds off the appendix.

A.1 PAM matrix derivation

The PAM series of mutation probability matrices were introduced by Dayhoff et al. in [DSO78]. PAM is an acronym for *Point Accepted Mutation*, which is a mutation (a substitution of amino acids in a protein) that is *accepted* by evolution in the sense that it has not been rejected by natural selection. PAM matrices are associated with a number which gives the number of mutations per 100 amino acids. For example, the PAM1 matrix contains the probabilities of mutation on a time interval of 1 mutation per 100 amino acids - that is, we expect 1% of the amino acids to undergo mutation. Similary, the probabilities in the PAM250 matrix applies to time intervals of 250 mutations per 100 amino acids.

Random and independent mutations are assumed. Thus, the mutation probability of a specific amino acid depends only on the amino acid itself, implying a Markovian model of evolution. The PAM matrices are therefore Markov chains.

All of the PAM matrices can be derived from the PAM1 matrix. To build this matrix, one needs a collection of *phylogenetic trees* representing the accepted mutations in families of sequences (i.e. proteins). A phylogenetic tree shows the evolutionary relationships among species or other biological entities which are assumed to have a common ancestor. Given a node in such a tree, the parent node represents the most recent common ancestor. Edge lengths, if given, correspond to some kind of time estimate.

Immediate mutations like $i \rightarrow j$, are assumed (not intermediate ones like $i \rightarrow k \rightarrow j$). To avoid a large degree of ambiguity, Dayhoff et al. therefore restricted the collection to sequence families with more than 85% identity. Using this collection, the number of changes from amino acid i to amino acid j, A_{ij}, is counted. The total number of changes into amino acid j can then easily be calculated as $C_j = \sum_{i=1}^{20} A_{ij}$. Using T_j to represent the number of appearences of j in the trees, the *relative mutability* of j is defined as shown in equation A.1.

$$m_j = \frac{C_j}{T_j} \tag{A.1}$$

The relative mutability of amino acid j is proportional to the probability of j changing into another amino acid, $P(j\,\text{changes}) = cm_j$, where c is a constant which depends on evolutionary time. Remembering that $P(A, B) = P(B|A)P(A)$, the probability of i changing into j can be found as shown in equation A.3. Equation A.2 gives the probability of j changing into i given that j was observed to change, which can be found using the amino acid change counts.

$$P(j\,\text{changes to}\,i\,|\,j\,\text{changes}) = \frac{A_{ij}}{C_j} = \frac{A_{ij}}{\sum_{l=1}^{20} A_{lj}} \tag{A.2}$$

$$P_{ij} = P(j\,\text{changes to}\,i) = cm_j \frac{A_{ij}}{C_j} = \frac{cm_j A_{ij}}{\sum_{l=1}^{20} A_{lj}} \tag{A.3}$$

The 400 P_{ij} values constitute the PAM1 probability matrix. Diagonal elements in this matrix can be seen as the probability that the amino acids do not change over the evolutionary distance of interest. They are calculated as shown in equation A.4.

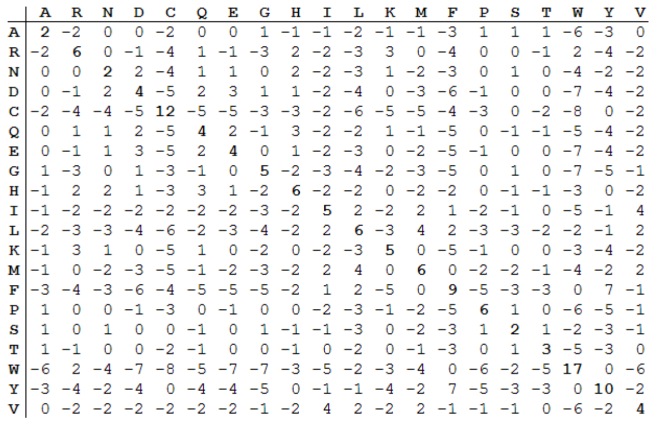
$$P(j\,\text{does not change}) = P_{jj} = 1 - cm_j \tag{A.4}$$

The value of c is chosen so that the average number of changes per 100 amino acids is 1. Given the relative frequency f_j of the amino acid j in the data set, it can be calculated as shown in equation A.5.

$$c = \frac{1}{100 \sum_{j=1}^{20} f_j m_j} \tag{A.5}$$

It turns out that all PAM matrices can be derived directly from PAM1 by simply multiplying the PAM1 matrix by itself the desired number of times. For example, the PAM250 probability matrix can be calculated by multiplying PAM1 by itself 250 times.

For two amino acids i and j paired in an alignment, P_{ij} gives the probability of i having changed into j due to mutation. But there is a problem with using the



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**2**	−2	0	0	−2	0	0	1	−1	−1	−2	−1	−1	−3	1	1	1	−6	−3	0
R	−2	**6**	0	−1	−4	1	−1	−3	2	−2	−3	3	0	−4	0	0	−1	2	−4	−2
N	0	0	**2**	2	−4	1	1	0	2	−2	−3	1	−2	−3	0	1	0	−4	−2	−2
D	0	−1	2	**4**	−5	2	3	1	1	−2	−4	0	−3	−6	−1	0	0	−7	−4	−2
C	−2	−4	−4	−5	**12**	−5	−5	−3	−3	−2	−6	−5	−5	−4	−3	0	−2	−8	0	−2
Q	0	1	1	2	−5	**4**	2	−1	3	−2	−2	1	−1	−5	0	−1	−1	−5	−4	−2
E	0	−1	1	3	−5	2	**4**	0	1	−2	−3	0	−2	−5	−1	0	0	−7	−4	−2
G	1	−3	0	1	−3	−1	0	**5**	−2	−3	−4	−2	−3	−5	0	1	0	−7	−5	−1
H	−1	2	2	1	−3	3	1	−2	**6**	−2	−2	0	−2	−2	0	−1	−1	−3	0	−2
I	−1	−2	−2	−2	−2	−2	−2	−3	−2	**5**	2	−2	2	1	−2	−1	0	−5	−1	4
L	−2	−3	−3	−4	−6	−2	−3	−4	−2	2	**6**	−3	4	2	−3	−3	−2	−2	−1	2
K	−1	3	1	0	−5	1	0	−2	0	−2	−3	**5**	0	−5	−1	0	0	−3	−4	−2
M	−1	0	−2	−3	−5	−1	−2	−3	−2	2	4	0	**6**	0	−2	−2	−1	−4	−2	2
F	−3	−4	−3	−6	−4	−5	−5	−5	−2	1	2	−5	0	**9**	−5	−3	−3	0	7	−1
P	1	0	0	−1	−3	0	−1	0	0	−2	−3	−1	−2	−5	**6**	1	0	−6	−5	−1
S	1	0	1	0	0	−1	0	1	−1	−1	−3	0	−2	−3	1	**2**	1	−2	−3	−1
T	1	−1	0	0	−2	−1	0	0	−1	0	−2	0	−1	−3	0	1	**3**	−5	−3	0
W	−6	2	−4	−7	−8	−5	−7	−7	−3	−5	−2	−3	−4	0	−6	−2	−5	**17**	0	−6
Y	−3	−4	−2	−4	0	−4	−4	−5	0	−1	−1	−4	−2	7	−5	−3	−3	0	**10**	−2
V	0	−2	−2	−2	−2	−2	−2	−1	−2	4	2	−2	2	−1	−1	−1	0	−6	−2	**4**

Figure A.1: PAM250 score matrix

mutation probabilities directly as alignment scores: We are not sure whether the pair *ij* was actually a mutation. We cannot simply assume that all pairs in an alignment are have a common evolutionary ancestor. Instead, given a pair *ij*, we are interested in knowing if *j* replaces *i* more frequently than it appears by chance. The probability that an amino acid *j* appears by chance in a sequence can be estimated as the relative frequency f_j of *j* in the data set. The ratio P_{ij}/f_j is called *likelihood* or *odds*. Given a mutation probability matrix *P*, the scoring matrix *S* can finally be calculated as shown in equation A.6.

$$S_{ij} = 10 \log_{10} \frac{P_{ij}}{f_j} \tag{A.6}$$

Thus, the scoring matrix distinguishes real mutations from chance alignments. When computing the score of an alignment, the relatedness odds of all pairs needs to be multiplied to give the final probability. Since alignment algorithms like the Smith-Waterman algorithm introduced in section 2.2.3 require the scores to be additive, the logarithm of the odds is taken in equation A.6. The quantity S_{ij} is often referred to as the similarity between amino acids *i* and *j*. An example of a commonly used matrix from the PAM family, the PAM250 score matrix, is shown in figure A.1.

A.2 BLOSUM matrix derivation

The BLOSUM (BLOcks SUbsitution Matrices) family of substitution matrices was introduced in [HH92]. The work was driven by the need to model protein sequences of lesser degree of divergence than the PAM family (recall from the previous subsection that these matrices are derived from proteins which are at least 85% similar). The matrices are derived from 2000 BLOCKS coming from over 500 protein families. BLOCKS are sequence regions in proteins which do

```
GPPGVGKTSIGKSIARALNR
GPPGVGKTSLASSIAKALNR
GPPGVGKTSIARSIARALNR
GPPGVGKTSLGRSIAEALGR
```

Figure A.2: Example of a BLOCK of four proteins

not contain any insertions or deletions. A BLOCK can be visualized by stacking the sequences on top of each other. An example is shown in figure A.2, which has been adapted from [Car05].

Given a BLOCK, the first step is to count all possible pairs of amino acids in each column. For a column with n amino acids, the number of such pairs for this column is $\frac{n(n-1)}{2}$. The frequency of the pair ij is denoted as f_{ij}. Note that $f_{ij} = f_{ji}$ since the pairs are undirected. For example, the column containing amino acids RKRE in figure A.2 contains one RR pair, two KR pairs, two ER pairs and one EK pair. Using these frequencies, the probability of occurrence can be calculated for each pair ij as shown in equation A.7.

$$q_{ij} = \frac{f_{ij}}{\sum_{i=1}^{20} \sum_{j=1}^{i} f_{ij}} \tag{A.7}$$

The probability of amino acid i occurring in any pair can then be calculated as shown in equation A.8.

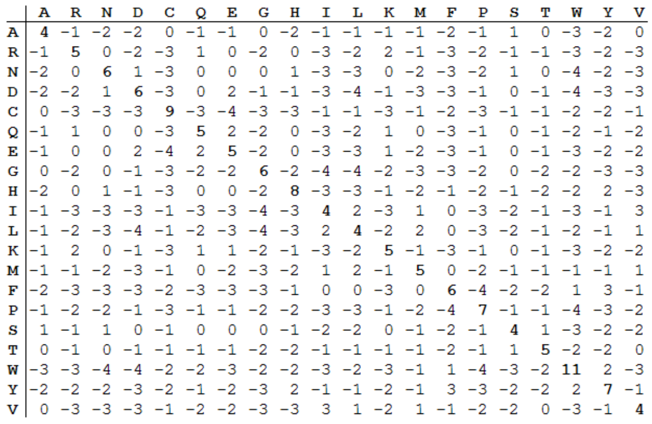
$$p_i = q_{ii} + \sum_{j \neq i} \frac{q_{ij}}{2} = q_{ii} + \sum_{j > i} q_{ij} \tag{A.8}$$

Assuming that the amino acids occur independently of each other in pairs, the expected probability of occurrence for each amino acid pair is calculated as shown in A.9.

$$e_{ij} = \begin{cases} p_i p_j & i = j \\ 2 p_i p_j & i \neq j \end{cases} \tag{A.9}$$

As with the PAM matrices, we are interested in the *odds* or *likelihood* of each amino acid pair. For a pair ij, this is the ratio between the probability that the transition $i \leftrightarrow j$ is a mutation and the probability that the pair occurs by random chance. The former is given by q_{ij} and the latter is given by e_{ij}, so the odds pairs are found as q_{ij}/e_{ij}. To make the scoring scheme additive, the logarithm of the odds is taken, so the final BLOSUM scoring matrix is found using equation A.10.

$$S_{ij} = 2 \log_2 \frac{q_{ij}}{e_{ij}} \tag{A.10}$$



	A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
A	**4**	−1	−2	−2	0	−1	−1	0	−2	−1	−1	−1	−1	−2	−1	1	0	−3	−2	0
R	−1	**5**	0	−2	−3	1	0	−2	0	−3	−2	2	−1	−3	−2	−1	−1	−3	−2	−3
N	−2	0	**6**	1	−3	0	0	0	1	−3	−3	0	−2	−3	−2	1	0	−4	−2	−3
D	−2	−2	1	**6**	−3	0	2	−1	−1	−3	−4	−1	−3	−3	−1	0	−1	−4	−3	−3
C	0	−3	−3	−3	**9**	−3	−4	−3	−3	−1	−1	−3	−1	−2	−3	−1	−1	−2	−2	−1
Q	−1	1	0	0	−3	**5**	2	−2	0	−3	−2	1	0	−3	−1	0	−1	−2	−1	−2
E	−1	0	0	2	−4	2	**5**	−2	0	−3	−3	1	−2	−3	−1	0	−1	−3	−2	−2
G	0	−2	0	−1	−3	−2	−2	**6**	−2	−4	−4	−2	−3	−3	−2	0	−2	−2	−3	−3
H	−2	0	1	−1	−3	0	0	−2	**8**	−3	−3	−1	−2	−1	−2	−1	−2	−2	2	−3
I	−1	−3	−3	−3	−1	−3	−3	−4	−3	**4**	2	−3	1	0	−3	−2	−1	−3	−1	3
L	−1	−2	−3	−4	−1	−2	−3	−4	−3	2	**4**	−2	2	0	−3	−2	−1	−2	−1	1
K	−1	2	0	−1	−3	1	1	−2	−1	−3	−2	**5**	−1	−3	−1	0	−1	−3	−2	−2
M	−1	−1	−2	−3	−1	0	−2	−3	−2	1	2	−1	**5**	0	−2	−1	−1	−1	−1	1
F	−2	−3	−3	−3	−2	−3	−3	−3	−1	0	0	−3	0	**6**	−4	−2	−2	1	3	−1
P	−1	−2	−2	−1	−3	−1	−1	−2	−2	−3	−3	−1	−2	−4	**7**	−1	−1	−4	−3	−2
S	1	−1	1	0	−1	0	0	0	−1	−2	−2	0	−1	−2	−1	**4**	1	−3	−2	−2
T	0	−1	0	−1	−1	−1	−1	−2	−2	−1	−1	−1	−1	−2	−1	1	**5**	−2	−2	0
W	−3	−3	−4	−4	−2	−2	−3	−2	−2	−3	−2	−3	−1	1	−4	−3	−2	**11**	2	−3
Y	−2	−2	−2	−3	−2	−1	−2	−3	2	−1	−1	−2	−1	3	−3	−2	−2	2	**7**	−1
V	0	−3	−3	−3	−1	−2	−2	−3	−3	3	1	−2	1	−1	−2	−2	0	−3	−1	**4**

Figure A.3: BLOSUM62 score matrix

Each BLOSUM matrix is associated with a number indicating the threshold of identity used when deriving the matrix. For example, the BLOSUM62 matrix was derived using sequences (BLOCKS) more than 62% identical. BLOCKS of less similarity were not considered. This matrix is included as an example of a matrix from the BLOSUM family in figure A.3.

A.3 A brief comparison of PAM and BLOSUM

The first difference to notice between the PAM and BLOSUM models is that PAM uses an explicit model of evolution, whereas BLOSUM uses an implicit model. Recall from the presentation of PAM that the PAM1 matrix is constructed from a pholygenetic tree which has been assembled by human domain experts. In this model, common evolutionary origin is specified explicitly. In contrast, evolutionary origin in BLOSUM matrices is implicit in the amino acid counts derived directly from highly conserved blocks - there is no explicit specification of common evolutionary ancestry.

Another key difference between the two models lies in the nature of the sequences used in the derivation process. The PAM model includes both highly conserved and highly mutated sequence regions, because the observed mutations are based on global rather than local alignments. In contrast, recall from the previous section that BLOSUM uses conserved regions in alignments where insertions or deletions do not occur (BLOCKS).

Empirical tests have shown that both the PAM and BLOSUM series of matrices perform well when used as scoring matrices in database searches [HH93]. However, as a local alignment scheme is often used in such searches, BLOSUM matrices are often found to perform slightly better. A general "rule of thumb" is to use BLOSUM matrices when searching for conserved domains in proteins regardless of evolutionary distance, and to use PAM matrices when searching for evolutionary origin.

A.4 Other amino acid substitution matrices

A number of alternative amino acid substitution matrices exist, differing in derivation method and dataset used. This subsection describes some of these briefly.

An approach different from the PAM and BLOSUM models is taken by Gonnet et al. in [GCB92]. In their work, exhaustive pairwise alignments of proteins in the entire database as it existed at that time are calculated. The results of these alignments are then used to estimate a distance matrix. The authors suggest that their matrix should be used instead of the PAM250 matrix. Refinements on the search results can thereafter be done using an appropriate matrix from the PAM family.

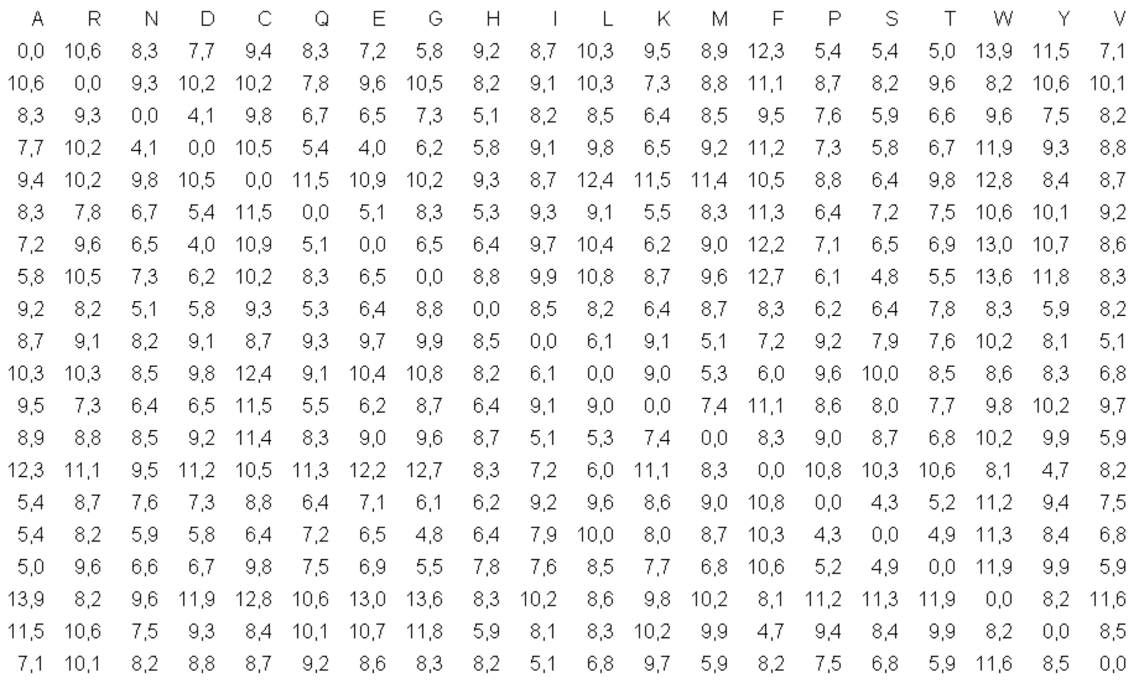
Yet another approach is taken by Risler et al. in [RDDH88] and Overington et al. in [ODJ$^+$92], where proteins are aligned based on three-dimensional structure rather than sequence. Therefore, only proteins for which a spatial structure has been determined are used. Substitution statistics are gathered from the resulting alignments. In theory, the substitution matrices generated using this method should perform better than both PAM, BLOSUM and Gonnet matrices. However, the reliability of the substitution statistics is questionable since a limited number of three-dimensional protein structures are available, so this has not proved to be the case in practice.

A problem with the PAM series of matrices derived in 1978 is that many of the possible amino acid substitutions were not observed in the dataset available. Jones et al. has constructed an updated version of the PAM250 mutation matrix, called PET91, in [JTT92]. This substitution matrix is based on 2621 families of sequences from the Swiss-Prot database[1], and covers many of the substitutions which were poorly represented in the original dataset.

[1]http://www.ebi.ac.uk/swissprot/

Appendix B

CCA-PAM70 matrix

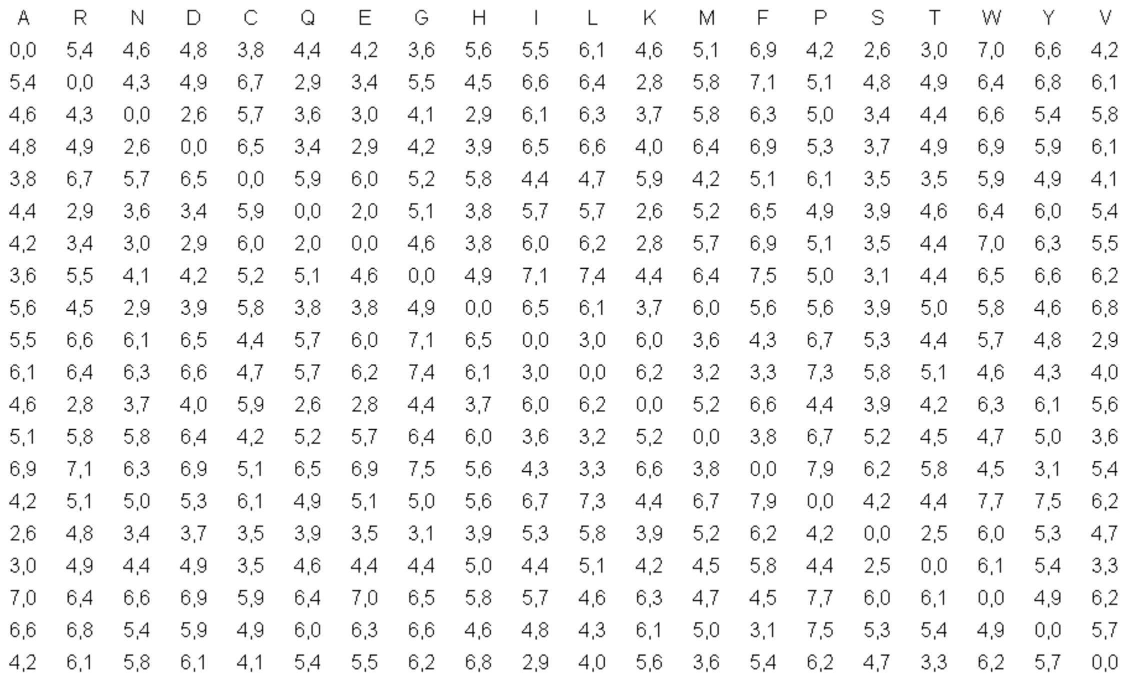


A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
0,0	10,6	8,3	7,7	9,4	8,3	7,2	5,8	9,2	8,7	10,3	9,5	8,9	12,3	5,4	5,4	5,0	13,9	11,5	7,1
10,6	0,0	9,3	10,2	10,2	7,8	9,6	10,5	8,2	9,1	10,3	7,3	8,8	11,1	8,7	8,2	9,6	8,2	10,6	10,1
8,3	9,3	0,0	4,1	9,8	6,7	6,5	7,3	5,1	8,2	8,5	6,4	8,5	9,5	7,6	5,9	6,6	9,6	7,5	8,2
7,7	10,2	4,1	0,0	10,5	5,4	4,0	6,2	5,8	9,1	9,8	6,5	9,2	11,2	7,3	5,8	6,7	11,9	9,3	8,8
9,4	10,2	9,8	10,5	0,0	11,5	10,9	10,2	9,3	8,7	12,4	11,5	11,4	10,5	8,8	6,4	9,8	12,8	8,4	8,7
8,3	7,8	6,7	5,4	11,5	0,0	5,1	8,3	5,3	9,3	9,1	5,5	8,3	11,3	6,4	7,2	7,5	10,6	10,1	9,2
7,2	9,6	6,5	4,0	10,9	5,1	0,0	6,5	6,4	9,7	10,4	6,2	9,0	12,2	7,1	6,5	6,9	13,0	10,7	8,6
5,8	10,5	7,3	6,2	10,2	8,3	6,5	0,0	8,8	9,9	10,8	8,7	9,6	12,7	6,1	4,8	5,5	13,6	11,8	8,3
9,2	8,2	5,1	5,8	9,3	5,3	6,4	8,8	0,0	8,5	8,2	6,4	8,7	8,3	6,2	6,4	7,8	8,3	5,9	8,2
8,7	9,1	8,2	9,1	8,7	9,3	9,7	9,9	8,5	0,0	6,1	9,1	5,1	7,2	9,2	7,9	7,6	10,2	8,1	5,1
10,3	10,3	8,5	9,8	12,4	9,1	10,4	10,8	8,2	6,1	0,0	9,0	5,3	6,0	9,6	10,0	8,5	8,6	8,3	6,8
9,5	7,3	6,4	6,5	11,5	5,5	6,2	8,7	6,4	9,1	9,0	0,0	7,4	11,1	8,6	8,0	7,7	9,8	10,2	9,7
8,9	8,8	8,5	9,2	11,4	8,3	9,0	9,6	8,7	5,1	5,3	7,4	0,0	8,3	9,0	8,7	6,8	10,2	9,9	5,9
12,3	11,1	9,5	11,2	10,5	11,3	12,2	12,7	8,3	7,2	6,0	11,1	8,3	0,0	10,8	10,3	10,6	8,1	4,7	8,2
5,4	8,7	7,6	7,3	8,8	6,4	7,1	6,1	6,2	9,2	9,6	8,6	9,0	10,8	0,0	4,3	5,2	11,2	9,4	7,5
5,4	8,2	5,9	5,8	6,4	7,2	6,5	4,8	6,4	7,9	10,0	8,0	8,7	10,3	4,3	0,0	4,9	11,3	8,4	6,8
5,0	9,6	6,6	6,7	9,8	7,5	6,9	5,5	7,8	7,6	8,5	7,7	6,8	10,6	5,2	4,9	0,0	11,9	9,9	5,9
13,9	8,2	9,6	11,9	12,8	10,6	13,0	13,6	8,3	10,2	8,6	9,8	10,2	8,1	11,2	11,3	11,9	0,0	8,2	11,6
11,5	10,6	7,5	9,3	8,4	10,1	10,7	11,8	5,9	8,1	8,3	10,2	9,9	4,7	9,4	8,4	9,9	8,2	0,0	8,5
7,1	10,1	8,2	8,8	8,7	9,2	8,6	8,3	8,2	5,1	6,8	9,7	5,9	8,2	7,5	6,8	5,9	11,6	8,5	0,0

Figure B.1: Derived CCA-PAM70 matrix, scaled by 0.05

Appendix C

GA-PAM250 matrix



A	R	N	D	C	Q	E	G	H	I	L	K	M	F	P	S	T	W	Y	V
0,0	5,4	4,6	4,8	3,8	4,4	4,2	3,6	5,6	5,5	6,1	4,6	5,1	6,9	4,2	2,6	3,0	7,0	6,6	4,2
5,4	0,0	4,3	4,9	6,7	2,9	3,4	5,5	4,5	6,6	6,4	2,8	5,8	7,1	5,1	4,8	4,9	6,4	6,8	6,1
4,6	4,3	0,0	2,6	5,7	3,6	3,0	4,1	2,9	6,1	6,3	3,7	5,8	6,3	5,0	3,4	4,4	6,6	5,4	5,8
4,8	4,9	2,6	0,0	6,5	3,4	2,9	4,2	3,9	6,5	6,6	4,0	6,4	6,9	5,3	3,7	4,9	6,9	5,9	6,1
3,8	6,7	5,7	6,5	0,0	5,9	6,0	5,2	5,8	4,4	4,7	5,9	4,2	5,1	6,1	3,5	3,5	5,9	4,9	4,1
4,4	2,9	3,6	3,4	5,9	0,0	2,0	5,1	3,8	5,7	5,7	2,6	5,2	6,5	4,9	3,9	4,6	6,4	6,0	5,4
4,2	3,4	3,0	2,9	6,0	2,0	0,0	4,6	3,8	6,0	6,2	2,8	5,7	6,9	5,1	3,5	4,4	7,0	6,3	5,5
3,6	5,5	4,1	4,2	5,2	5,1	4,6	0,0	4,9	7,1	7,4	4,4	6,4	7,5	5,0	3,1	4,4	6,5	6,6	6,2
5,6	4,5	2,9	3,9	5,8	3,8	3,8	4,9	0,0	6,5	6,1	3,7	6,0	5,6	5,6	3,9	5,0	5,8	4,6	6,8
5,5	6,6	6,1	6,5	4,4	5,7	6,0	7,1	6,5	0,0	3,0	6,0	3,6	4,3	6,7	5,3	4,4	5,7	4,8	2,9
6,1	6,4	6,3	6,6	4,7	5,7	6,2	7,4	6,1	3,0	0,0	6,2	3,2	3,3	7,3	5,8	5,1	4,6	4,3	4,0
4,6	2,8	3,7	4,0	5,9	2,6	2,8	4,4	3,7	6,0	6,2	0,0	5,2	6,6	4,4	3,9	4,2	6,3	6,1	5,6
5,1	5,8	5,8	6,4	4,2	5,2	5,7	6,4	6,0	3,6	3,2	5,2	0,0	3,8	6,7	5,2	4,5	4,7	5,0	3,6
6,9	7,1	6,3	6,9	5,1	6,5	6,9	7,5	5,6	4,3	3,3	6,6	3,8	0,0	7,9	6,2	5,8	4,5	3,1	5,4
4,2	5,1	5,0	5,3	6,1	4,9	5,1	5,0	5,6	6,7	7,3	4,4	6,7	7,9	0,0	4,2	4,4	7,7	7,5	6,2
2,6	4,8	3,4	3,7	3,5	3,9	3,5	3,1	3,9	5,3	5,8	3,9	5,2	6,2	4,2	0,0	2,5	6,0	5,3	4,7
3,0	4,9	4,4	4,9	3,5	4,6	4,4	4,4	5,0	4,4	5,1	4,2	4,5	5,8	4,4	2,5	0,0	6,1	5,4	3,3
7,0	6,4	6,6	6,9	5,9	6,4	7,0	6,5	5,8	5,7	4,6	6,3	4,7	4,5	7,7	6,0	6,1	0,0	4,9	6,2
6,6	6,8	5,4	5,9	4,9	6,0	6,3	6,6	4,6	4,8	4,3	6,1	5,0	3,1	7,5	5,3	5,4	4,9	0,0	5,7
4,2	6,1	5,8	6,1	4,1	5,4	5,5	6,2	6,8	2,9	4,0	5,6	3,6	5,4	6,2	4,7	3,3	6,2	5,7	0,0

Figure C.1: Derived GA-PAM250 matrix, scaled by 0.1

Bibliography

[AASK04a] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. Boostmap: A method for efficient approximate similarity ranknings. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2004.

[AASK04b] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. Learning euclidean embeddings for indexing and classification. Technical Report 2004-014, Boston University, April 2004.

[AGM$^+$90] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *J. Mol. Biol.*, 215(3):403–410, 1990.

[BBM93a] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 1, fundamentals. *University Computing*, 15(2):58–69, 1993.

[BBM93b] D. Beasley, D. R. Bull, and R. R. Martin. An overview of genetic algorithms: Part 2, research topics. *University Computing*, 15(4):170–181, 1993.

[BFM$^+$96] J. E. Barros, J. French, W. Martin, P. M. Kelly, and T. M. Cannon. Using the triangle inequality to reduce the number of comparisons required for similarity-based retrieval. In *SPIE Vol. 2670 Storage and Retrieval for Still Image and Video Databases IV*, pages 392–403. 1996.

[BK73] W. A. Burkhard and R. M. Keller. Some approaches to best-match file searching. *Commun. ACM*, 16(4):230–236, 1973.

[BO97] T. Bozkaya and M. Ozsoyoglu. Distance-based indexing for high-dimensional metric spaces. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 357–368, 1997.

[BY97] R. Baeza-Yates. *Searching: An algorithmic tour*, volume 37, pages 331–359. Marcel Dekker Inc., 1997.

[BYCMW94] R. Baeza-Yates, G. Cunto, U. Manber, and S. Wu. Proximity matching using fixed-queries trees. In *Proc. 5th Combinatorial Pattern Matching (CPM'94)*, pages 192–212, 1994.

[Car05] M. Caron. The bioinformatics knowledge base: BLOCKS, 2005. `http://apps.bioneq.qc.ca/twiki/bin/view/Knowledgebase/BLOCKS`. Cited May 15th 2005.

[Chi94] T. Chieueh. Content-based image indexing. In *Proc. 20th Conference on Very Large Databases (VLDB'94)*, pages 582–593, 1994.

[CNBYM99] C. Chávez, E. Navarro, G. Baeza-Yates, and R. Marroquín. Searching in metric spaces. Technical Report TR/DCC-99-3, University of Chile, Department of Computer Science, 1999.

[DSO78] M. O. Dayhoff, R. M. Schwartz, and B. C. Orcutt. A model of evolutionary change in proteins. In *Atlas of protein sequence and structure, supplement 3*, pages 345–352. National Biomedical Research Foundation, Washington DC, 1978.

[DST04] I. S. Dhillon, S. Sra, and J. A. Tropp. Triangle fixing algorithms for the metric nearness problem. Technical Report TR-04-22, University of Texas, Austin, Department of Computer Sciences, September 2004.

[Dyk83] R. L. Dykstra. An algorithm for restricted least squares regression. *Journal of the American Statistical Association*, 78(384):837–842, 1983.

[FL95] C. Faloutsos and K. I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, California, 22–25 1995.

[FS97] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[GB99] W. N. Grundy and T. L. Bailey. Family pairwise search with embedded motif models. *Bioinformatics*, 15(6):463–470, 1999.

[GCB92] G. H. Gonnet, M. A. Cohen, and S. A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256(5062):1443–1445, June 1992.

[GCS70] P. E. Green, F. J. Carmone, and S. M. Smith. *Multidimensional scaling: concepts and applications*. Allyn and Bacon, Boston, 1970.

[GD91] M. Gribskov and J. Devereux. *Sequence Analysis Primer*, chapter 3. Stockton Press, New York, 1991.

[GKM03] P. Gupta, A. B. Kahng, and S. Mantik. Routing-aware scan chain ordering. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 857–862, January 2003.

[Got82] O. Gotoh. An improved algorithm for matching biological sequences. *J. Mol. Biol.*, 162:705–708, 1982.

[GR96] M. Gribskov and M. L. Robinson. Use of receiver operating characteristic (ROC) analysis to evaluate sequence matching. *Computers & Chemistry*, 20(1):25–33, 1996.

[HBK$^+$03] E. Halpering, J. Buhler, R. Karp, R. Krauthgamer, and B. Westover. Detecting protein sequence conservation via metric embeddings. *Bioinformatics*, 19(Suppl. 1):i122–i129, 2003.

[HH92] S. Henikoff and J. G. Henikoff. Amino acid substitution matrices from protein blocks. In *Proc. Nat. Acad. Sci USA*, pages 10915–10919, November 1992.

[HH93] S. Henikoff and J. G. Henikoff. Performance evaluation of amino acid substitution matrices. *Proteins*, 17(1):49–61, 1993.

[HHLB04] Y. Hou, W. Hsu, M. L. Lee, and C. Bystroff. Remote homology detection using local sequence-structure correlations. *Proteins*, 57(3):518–530, 2004.

[HS03] G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(5):530–549, 2003.

[JTT92] D. T. Jones, W. R. Taylor, and J. M. Thornton. The rapid generation of mutation data matrices from protein sequences. *Comput. Appl. Biosci.*, 8:275–282, 1992.

[LLTY97] M. Linial, N. Linial, N. Tishby, and G. Yona. Global self organization of all known protein sequences reveals inherent biological signatures. *J. Mol. Biol.*, 268(2):539–556, 1997.

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

[NW70] S. B. Needleman and C. D. Wunsch. A general method applicable to search for similarities in the amnio acid sequence of proteins. *J. Mol. Biol.*, 48(3):443–453, March 1970.

[ODJ$^+$92] J. Overington, D. Donnelly, M. S. Johnson, A. Sali, and T. L. Blundell. Environment-specific amino acid substitution tables: Tertiary templates and prediction of protein folds. *Protein Sci.*, 1(2):216–226, Feb 1992.

[Par98] J. Paredis. Coevolutionary algorithms. In T. Bäck, D. Fogel, and Z. Michalewicz, editors, *The Handbook of Evolutionary Computation, 1st supplement*. Oxford University Press, 1998.

[RDDH88] J. L. Risler, M. O. Delorme, H. Delacroix, and A. Henaut. Amino acid substitutions in structurally related proteins. A pattern recognition approach. *J. Mol. Biol.*, 204:1019–1029, 1988.

[SAM$^+$01] A. A. Schäffer, L. Aravind, T. L. Madden, S. Shavirin, J. L. Spouge, Y. I. Wolf, E. V. Koonin, and S. F. Altschul. Improving the accuracy of PSI-BLAST protein database searches with composition-based statistics and other refinements. *Nucleid Acids Research*, 29(14):2994–3005, 2001.

[SS99] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidece-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[SW81] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *J. Mol. Biol.*, 147:195–197, 1981.

[TJ93] W. R. Taylor and D. T. Jones. Deriving an amino acid distance matrix. *J. Theor. Biol.*, 164:65–83, 1993.

[TO04] Y. H. Taguchi and Y. Oono. Novel non-metric MDS algorithm with confidence level test, 2004. Published at `http://www.granular.com/MDS/src/paper.pdf`. Cited March 3rd 2005.

[TO05] Y. H. Taguchi and Y. Oono. Relational patterns of gene expression via non-metric multidimensional scaling analysis. *Bioinformatics*, 21(6):730–740, 2005.

[Wat95] M. S. Waterman. *Introduction to Computational Biology: Maps, Sequences and Genomes*. CRC Press, 1st edition edition, 1995.

[WLDJ01] R. P. Wiegand, W. C. Liles, and K. A. De Jong. An empirical analysis of collaboration methods in cooperative coevolutionary algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 1235–1245, 2001.

[WMMY02] R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye. *Probability and statistics for Engineers and Scientists, Seventh edition*, chapter 16, pages 601–605. Prentice Hall, Inc., Upper Saddle River, New Jersey, 2002.

[XM04] W. Xu and D.P. Miranker. A metric model of amino acid substitution. *Bioinformatics*, 20(8):1214–1221, 2004.

[XNJR03] E. Xing, A. Ng, M. Jordan, and S. Russell. Distance metric learning, with application to clustering with side-information. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in neural information processing systems 15*, pages 505–512. MIT Press, Cambridge, MA, USA, 2003.

[Yia93] P. Yianilos. Data structures and algorithms for nearest neighbour search in general metric spaces. In *Proc. 4th ACM-SIAM Symposium on Discrete Algorithms (SODA'93)*, pages 311–321, 1993.

[YJF98] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient retrieval of similar time sequences under time warping. In *Proceedings of the 14th International Conference on Data Engineering (ICDE'98)*, pages 201–208, 1998.