

Pattern Discovery in Biology: Theory and Applications

by

Aristidis Floratos

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
Department of Computer Science
New York University
January 1999

Approved: _____

Ravi Boppana

Approved: _____

Isidore Rigoutsos

© Aristidis Floratos
All Rights Reserved, 1999

This thesis is dedicated to everybody will ever invest the time to read it. It was (literally) written for you . . .

Acknowledgements

It was the January of 1993 when I first came to New York City and NYU. I still remember very vividly the first thing that made an impression on me: it was snowing like hell. Coming from a country with a mild climate, I had never seen an entire city painted white. Yet, unfamiliar as the terrain was, I could not help but feeling a rush of excitement under my skin. The city was buzzing with life and promise. Six years later, I stand at the end of yet another personal milestone, that of completing my graduate studies. These have been six interesting years, full of challenges, new people and new experiences. It was not always easy. I had to adjust from a small town environment and code of living to the sometimes intimidating fluidity of situations and characters in this vibrant metropolis. In the process, I have grown (and still am) both as a person and as a scientist. Amidst all these changes, though, there still is one feeling that remains the same as the first day I came here: I am still excited. For that, I am thankful.

There are two groups of people that I would like to thank: those that had something to do with my thesis and those with no relation what so ever.

Starting with the first group, I extend my gratitude to NYU and the Computer Science department for supporting me financially all these years. Without this support I would probably not have been here in the first place. I would also like to thank all the faculty members that I took a class with for providing stimulating courses and for sharing their expertise and insights, as well as for always being available to answer even my most naive questions. Thank you all guys.

I would also like to recognize the support that I got from IBM Research. The work presented in this dissertation started and finished there. Apart from providing

the infrastructure needed for my research, IBM also offered me an intellectually stimulating environment in which to perform this research. Working there during the last 2 and a half years I had the opportunity to interact with several talented people. I am thankful to all of them for sharing their ideas and visions with me.

At an individual level, there are several people that I had the pleasure and the privilege to work with and learn from. First, my advisor Ravi Boppana. Ravi has been a delight to work with, coupling his deep understanding of theoretical issues with a human kindness and modesty that is extremely rare in an environment where overblown egos sometimes abound. I am thankful to him for sharing his ideas and experience, for teaching me how to think analytically, for always providing insightful directions for work and for never complaining, even when I was less than productive.

My second advisor, Isidore Rigoutsos, introduced me to the area of Computational Biology in 1996, when I first went to work as a summer student at IBM's T.J.Watson Research Center. He was the one who drew my attention to the many applications of pattern discovery in the context of Biology. For the past two years and a half, we have had a very prolific cooperation. During this time, I have come to appreciate his innovative thinking, his ability to quickly concentrate on the most crucial aspects of a problem, his knack for identifying important problems as well as his focus and dedication to whatever problem we are working on.

Paul Spirakis, my advisor in Greece, has been both a good friend and a source of inspiration. I am grateful to him for being my teacher, for sharing with me his enthusiasm about Science, for all the exciting and funny discussions we have had over the years, for always being there to listen to me and for encouraging me to pursue graduate studies in the US.

I would also like to thank Davi Geiger, Ken Perlin and Ben Goldberg for accepting to be members of my committee. Special thanks to Ben not only for teaching me one of the most interesting classes I've ever taken (his Honors Compilers) but also for showing the value of the lobe in squash ...

There are also a lot of other people with whom our paths crossed while all at NYU. With Laxmi Parida and Saugata Basu I shared offices and had the pleasure

of knowing them both as friends and as scientists. With Arash Baratloo and Peter Wyckoff we have traded ideas, jokes and occasionally smokes.

Regarding the second group of people (those that had nothing to do with this thesis), I would like to start from my family. They have always been there for me, giving me their unconditional love and supporting me in all my decisions. There are no words to express my gratitude to them. Their affection has been a source of energy for me both during good and bad times.

I consider myself lucky because I have been blessed with good friends. With my college buddies, Kostas Pantazopoulos, Dimitris Kehagias, Alex Terzis, Yannis Apostolidis and Dimitris Spartiotis, we all found ourselves in the US either for studies or for work. Having them around, made coping with things so much easier. Our common experiences and mutual support has helped our bond grow stronger and I am sure that it will stay like that for years to come. I am also thankful to Yiorgos Kaouris, Iosif Iosifidis and Christos Papadimakis for remaining good friends even after all these years that we have been apart. Whenever I return home for a few days of vacation they make me feel as if I never left.

Contents

Dedication	iv
Acknowledgements	v
List of Figures	xi
1 Introduction	1
1.1 Human Genome Project: History and Perspective	1
1.2 Scope of the thesis	4
1.3 Contributions	7
1.4 A Biology Primer	9
1.4.1 Introduction	10
1.4.2 Cells and Heredity	11
1.4.3 History	29
1.5 Biological Data Repositories	36
1.6 Research Portfolio	39
2 Pattern Discovery Algorithms	60
2.1 Problem Description	60
2.2 Previous Work	65
2.2.1 Alignment Algorithms	66
2.2.2 Pattern Enumeration Algorithms	69
2.2.3 Conclusions	75
2.3 TEIRESIAS	76

2.3.1	Terminology and Problem Definition	77
2.3.2	Hardness	78
2.3.3	Implementation	80
2.3.4	Correctness	83
2.3.5	Time Complexity	85
2.3.6	Experimental Performance	91
2.4	Results	93
2.4.1	Core Histones H3 and H4	93
2.4.2	Leghemoglobins	97
2.5	Discussion	100
2.6	Pattern Discovery Revisited	102
3	Large Scale Pattern Discovery	124
3.1	Introduction	125
3.2	Why Seqlets?	126
3.2.1	Seqlet Vistas In The Sequence Space	128
3.2.2	The Biology of Seqlets	130
3.3	Methodology	132
3.3.1	Treating Redundancy	132
3.3.2	Low Complexity Regions	133
3.3.3	Seqlet Statistics	134
3.4	Results	139
3.4.1	The three data bases	141
3.5	Seqlets and Clustering	143
4	Sequence Homology Detection	164
4.1	Introduction	164
4.2	Existing Methods	167
4.2.1	FASTA	167
4.2.2	BLAST	170
4.2.3	BLOCKS	173
4.3	Motivation and Definitions	175

4.4	Implementation	176
4.4.1	Searching	177
4.5	Results	180
4.5.1	Information Gathering	181
4.5.2	Searching	183
4.6	Discussion	189
5	Other Applications And Future Work	202
5.1	Multiple Sequence Alignment	202
5.2	Computer Security	208
5.3	Future Work	213
5.3.1	Improvements in TEIRESIAS	214
5.3.2	Validation of Seqlets	216
	Bibliography	227

List of Figures

1.1	Accumulation of data in the GenBank database	4
1.2	Evolution of a primordial protein into its present-day instances . . .	42
1.3	Amino acids and formation of proteins via peptide bonds.	43
1.4	The amino acids Alanine and Leucine.	44
1.5	Formation of the DNA polymeric chain.	46
1.6	Three dimensional representation of the DNA double helix.	47
1.7	The genetic code.	48
1.8	Coding and non-coding regions on a DNA molecule.	49
1.9	DNA trascription.	50
1.10	Translation of mRNA to protein.	51
1.11	Transcription and Translation.	52
1.12	Replication of the DNA double helix.	53
1.13	Sequencing a long DNA molecule.	54
1.14	Example of a genetic map.	55
1.15	Three dimensional structure of protein backbone.	56
1.16	An α -helix.	57
1.17	Primary and tertiary structure of a hemoglobin protein.	58
1.18	Profile building from a set of homologous protein regions.	59
2.1	Generating a pattern from aligned sequences	108
2.2	Example of a multiple sequence alignment.	109
2.3	Multiple sequence alignment of histone sequences.	110
2.4	The domain-swap problem in multiple sequence alignment	111

2.5	List of offsets generated by TEIRESIAS.	112
2.6	A convolution operation used by TEIRESIAS.	113
2.7	<i>Prefix-wise</i> and <i>suffix-wise</i> orderings.	114
2.8	Formal definition of the prefix-wise and suffix-wise relations.	115
2.9	Pseudo-code for the TEIRESIAS algorithm.	116
2.10	Generating patterns from a template and a sequence.	117
2.11	Experimental running times; $W = 10$	118
2.12	Experimental running times; $W = 15$	119
2.13	Patterns shared by both Histone H3 and H4 sequences.	120
2.13	(Continued from last page.) Patterns common to proteins of both the H3 and the H4 core histone families.	121
2.14	Patterns belonging to either the H3 or the H4 histone family.	122
2.15	Patterns belonging to the Leghemoglobin family.	123
3.1	Multidimensional scaling protein regions containing the seqlet “KP- KLGL”.	149
3.2	Example of domain-sharing by proteins.	150
3.3	Redundant group with a multidomain leader.	151
3.4	A protein with a low complexity region.	151
3.5	Chemotaxis proteins containing the seqlet “ALNAA..AA..A”.	152
3.6	Pattern Distribution in the cleaned-up version of SwissProt vs. Random.	153
3.7	Coverage of the “cleaned-up” data base of the 6 genomes.	155
3.8	Coverage of the “cleaned-up” SwissProt Rel. 34.	156
3.9	Coverage of the “cleaned-up” non-redundant data base.	157
3.10	Amino acid composition of most frequent patterns in the non re- dundant data base.	158
3.11	Example of ambiguous clustering.	161
3.12	Using seqlets to cluster proteins	162
3.13	Intersecting seqlets and the respective offset lists.	163
4.1	Example FASTA table for two sequences.	168

4.2	Distributions of FASTA scores.	191
4.3	Block defined by a generator pattern in BLOCKS.	192
4.4	Sample hash table for query sequence.	193
4.5	Segment chaining.	194
4.6	Distribution of SwissProt patterns characteristics.	195
4.7	Homology search for the protein <i>H31_HUMAN</i>	196
4.8	Homology search for the protein <i>YZ28_METJA</i>	197
4.9	Homologies induced by the kinase pattern “Y..S..I..DLK”	198
4.10	Distribution of patterns on <i>YZ28_METJA</i>	199
4.11	Pattern-induced alignments using the SwissProt annotation.	200
4.12	Characterization of regions of <i>YZ28_METJA</i> by using the Swiss- Prot annotation.	201
5.1	Alignment conflicts induced by patterns.	220
5.2	A consistent ordering of patterns.	221
5.3	The architecture of the intrusion detection system.	222
5.4	Translating a set of sequences using a reduced alphabet.	223
5.5	Problem with the convolution of bracketed expressions.	224
5.6	Distribution of RMS errors.	225
5.7	Alignment of backbones corresponding to a seqlet.	226

Chapter 1

Introduction

Molecular Biology studies the composition and interactions of life's agents, namely the various molecules (e.g. DNA, proteins, lipids) sustaining the living process. Traditionally, this study has been performed in wet labs using mostly physico-chemical techniques. Such techniques, although precise and detailed, are often cumbersome and time consuming. On top of that, recent advances in sequencing technology have allowed the rapid accumulation of DNA and protein data. As a result a gap has been created (and is constantly being expanded): on the one side there is a rapidly growing collection of data containing all the information upon which life is built; and on the other side we are currently unable to keep up with the study of this data, impaired by the limits of existing analysis tools. It is obvious that alternative analysis techniques are badly needed.

In this work we examine how computational methods can help in drilling the information contained in collections of biological data. In particular, we investigate how sequence similarity among various macromolecules (e.g. proteins) can be exploited towards the extraction of biologically useful information.

1.1 Human Genome Project: History and Perspective

(The reader is advised to start by skimming through the Biology Primer of Section 1.4. Through out the text there will be references to definitions and concepts

introduced therein.)

The *Human Genome Project (HGP)* was officially launched in the United States on October 1 1990, under the joint control of the department of Energy and the National Institutes of Health. The goal of the project, in the words of the National Human Genome Research Institute (<http://www.nhgri.nih.gov/HGP/>) is:

“...to construct detailed genetic and physical maps of the human genome, to determine the complete nucleotide sequence of human DNA, to localize the estimated 50,000-100,000 genes within the human genome, and to perform similar analyses on the genomes of several other organisms used extensively in research laboratories as model systems.”

Soon after the launch of the US program the effort became international in scope with several other countries (United Kingdom, France, Japan, Canada e.t.c.) joining in with similar, government-funded projects. The original schedule called for the completion of the human genome sequencing by the year 2005. Recently [23], the schedule has been revised and now the year 2003 is projected as the end date for the project.

The HGP is mostly concerned with the task of *sequencing* i.e. obtaining the sequence of nucleotides comprising selected genomes. Its goals are restricted in the construction of high-resolution *genetic* and *physical* maps of the organisms studied. This process, however, is well understood to be only the first step in the analysis and study of the organization of the genome (a research endeavor which has become known as *Genomics*). The real value is hidden in understanding the biological function of the various macromolecules (proteins and DNA) within their host organisms. In particular, there is a number of questions that can be asked:

- Under what conditions does a gene express its corresponding protein?
- What is the function of a protein within the cell? How does it interact with other molecules and what are the results of this interaction?
- What is the result of a mutation to a gene? How does this affect the operation of the protein the gene encodes? How does the *phenotype* of the host organism

change (if at all)?

- What is the role of the *control regions* on DNA in the expressibility of neighboring genes.
- Which genes (or combinations of genes) are related to diseases?
- What is the 3-dimensional structure of a protein? Is there a way to predict it directly from its sequence?
- Are there any basic building blocks of amino acids reused in the construction of proteins? Which are they? Is it possible to put them together in order to build new proteins?

Answering these questions using traditional biochemical techniques is a very time consuming proposition. For example, the straightforward way to determine the 3-dimensional structure of a protein involves the use of physical methods like *x-ray crystallography* or (for smaller molecules) *NMR spectroscopy*. Such methods can take months, even years, to produce the desired results. Furthermore, due to physical constraints, these techniques are not applicable to all types of proteins (for details, see chapter 17 in [17]).

The inherent difficulty of applying physicochemical analysis methods in conjunction with the exponential increase in the size of genomic databases (Fig. 1.1) necessitates the development of new techniques which will allow the rapid study of the above questions. The invention and application of such techniques to the study of the functional and structural properties of genes has been termed *functional genomics* (in contrast with the technologies developed for the sequencing of the genome and which are collectively referred to as *structural genomics*). Computer science is expected to play a key role in this new area, much more than just provide the tools for organizing and disseminating the sequencing data; as Hieter and Boguski put it in [48]:

“Computational biology will perform a critical and expanding role in this area [of functional genomics]: whereas structural genomics has been

characterized by data management, functional genomics will be characterized by mining the data sets for particularly valuable information.”

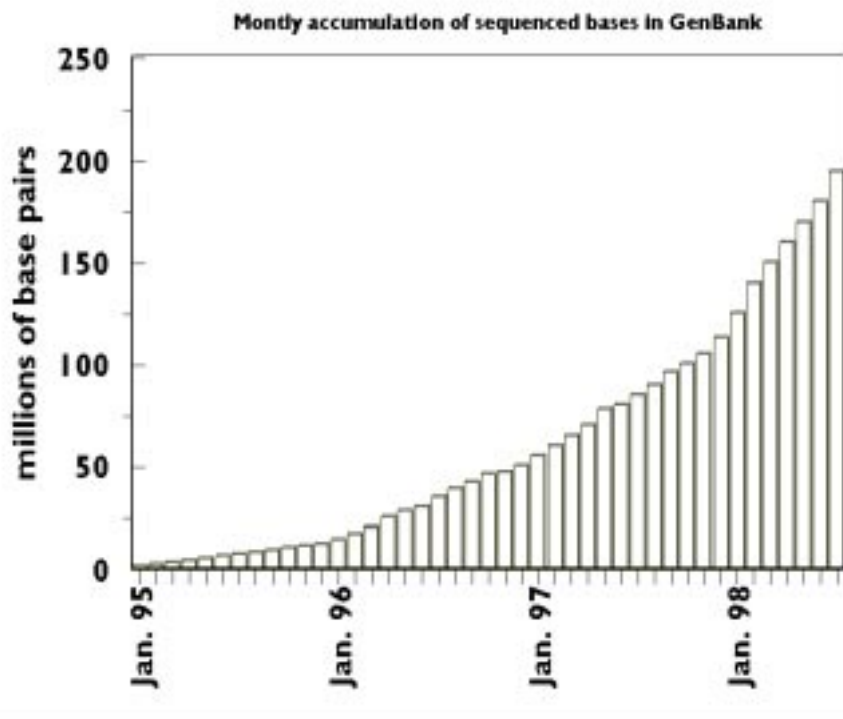


Figure 1.1: Yearly increase of the size (in millions of sequenced bases) of the GenBank database (<http://www.ncbi.nlm.nih.gov/>).

1.2 Scope of the thesis

In the spirit of the above quote, this work examines how mining massive data sets of genetic sequences can reveal hidden information of substantial biological importance. What we will be looking for (the object of the mining process) is *sequence* similarities, i.e. stretches of residues which are shared by several different

sequences. Similarities of an unexpected nature are then used to draw relationships between the sequences containing them.

It is interesting to notice that our data will almost invariably be just strings of characters (the 1-dimensional representation of biosequences); this is almost an oxymoron as DNA and proteins are organic entities and it seems hardly possible that they could be studied without taking into account their chemical properties. Fortunately, this is not the case. In what Gusfield [41] refers to as the *first fact of biological sequence analysis*:

“In biomolecular sequences (DNA, RNA or amino acid sequences), high sequence similarity usually implies significant functional or structural similarity.”

This fact allows the introduction of analysis methodologies that exploit the sequence homology between compared biosequences in order to infer useful information about them. The underlying principle here is that of “information transfer”. If, for example, a protein Q under examination is *sufficiently* similar to another protein A of known functionality then by using the fact quoted above, one can *transfer* the functionality of A to Q and guess that Q must also play a related (if not the same) biological role (this is actually the idea behind a number of very popular homology detection tools such as BLAST [3, 4] and FASTA [72, 84]). The exact notion of “sufficient similarity” varies between different methods of sequence comparison but is invariably defined in terms of the edit distance (using appropriate cost matrices) between Q and A and the statistical distribution of this distance [72, 61, 60].

Other methods [112, 50, 98, 51, 79, 96, 87, 95, 57, 77, 89, 108, 117, 103, 90] exploit the first fact of biological sequence analysis in a different manner: they start with several proteins from the same family (e.g. all the hemoglobin proteins from different mammals) and try to identify *sequence signatures* diagnostic of that protein family. A sequence signature is usually a restricted regular expression matching all (or most) of the proteins in the given family: we will be referring to such regular expressions as *patterns*. In other cases, weighted matrices of amino

acids (called *profiles*) [40, 39] or *hidden Markoff models (HMMs)* [64] are used as signatures. After a signature is derived it is used as a diagnostic device: when a new sequence is checked for membership in the protein family at hand, one checks if the new sequence contains some substring described by the signature for that family.

In order to appreciate why sequence signatures and in particular patterns are a good way to represent features characteristic of families of proteins it is helpful to take a look in the way that nature works. The dominant theory in Biology today sustains that the species evolve by *diversification*. This process occurs through mutations accumulated on the DNA of individual members of a species and the inheritance of such mutations from one generation to another. Mutations in coding regions of the DNA propagate (through the transcription/translation mechanism) to the proteins that the regions code for. As a result, it is possible to describe the evolutionary history of a given protein strain as a tree (Fig. 1.2). At the leaves of that tree we find all the proteins of the given strain surviving today across various organisms; these are the proteins forming a protein family like the hemoglobin proteins mentioned above. The root of the tree is occupied by that primordial protein (the *mother protein*) which gave rise to all the proteins of the tree. Intermediate nodes contain the various forms of the protein through evolution. The branches emanating from a node indicate diversification events where mutations gave rise to new proteins (the children of the node)¹. The result of this process (Fig 1.2) is an ability to find remnants of the mother protein in the descendant sequences. This is made possible by another biological fact: Specifically, although mutations happen randomly over the entire length of a protein, the results of individual mutations can vary wildly. This happens because not all the areas of a protein are created equal. Roughly speaking, a protein can be decomposed into

¹*Evolutionary trees* such as that in Fig. 1.2 can only be guessed since the only data available are the sequences at the leaves of the tree. Formally, the reconstruction takes the form of a minimization problem. What we seek to minimize is an objective function describing the overall mutation cost of the tree. For a detailed discussion of the problem the reader can look at chapter 17 of [41] and chapter 14 of [111]

two types of sub-regions: those which are vital for the operation of the protein (e.g. active sites of catalytic activity) and those which play more of a supporting role. The latter are much more tolerant to mutations than the former. In other words, mutations that occur in the vital regions of the protein are more likely to result in the death of the resulting offspring, thus resisting the modification of the original sequence. Consequently, preservation of the mother protein in these regions is much stronger (and detectable) than in the less important areas of the protein. It is exactly these conserved regions that the notion of a sequence signature intends to capture. Because patterns are capable of modeling both conserved amino acid positions and sites of mutations (*don't care* regions) they have been used extensively as representations of sequence signatures [9, 46, 6].

1.3 Contributions

The importance of patterns in the analysis of biological sequences was the original motivation behind the work presented in this thesis. Discovering such patterns in families of related proteins has been the centerpiece of several important databases of signatures (e.g. the PROSITE database [9], the PRINTS database [6], the BLOCKS database [46]) which play a central role in the every day practice of molecular biology.

Originally, the small amount of available data made it possible to obtain patterns for protein families by manual alignment and inspection (this was the process used originally in generating the PROSITE database). However, as more and more sequences come out of the sequencing machines, this approach is becoming inefficient. Tools for the automatic discovery of patterns are needed. Unfortunately, for any reasonable definition of a pattern it turns out that the relevant discovery problem is NP-hard. As a result, existing algorithms take one of two approaches: either they settle for incomplete results (approximation algorithms) in order to achieve reasonable performance, or, their running times render them impractical for even medium-sized inputs.

In this work we present TEIRESIAS, a new combinatorial algorithm for the

pattern discovery problem with a number of desirable features:

1. It only focuses on *maximal* patterns. Non-maximal patterns are redundant in that they offer no new knowledge and thus it makes no sense spending time to compute them. Computing such patterns is, as we are going to discuss, one of the main reasons for the horrible performance of exact pattern discovery algorithms.
2. It introduces a novel concept of *density* in the definition of patterns, thus resulting in biologically more relevant patterns.
3. It is general in nature and can be used for the discovery of patterns in any domain where the data can be represented as strings of characters. As such, it can be thought as a general purpose data mining algorithm.
4. Experimental results show that the algorithm is *output sensitive* i.e. its running time is quasi-linear to the size of the generated output. This property is very important for algorithms that solve hard problems exactly because it guarantees that these algorithms do not waste time in unnecessary computations.

The last of the algorithm's properties (i.e. its output sensitivity) makes it particularly appropriate for large inputs. This was a design requirement as our intention right from the beginning was to be able to use this tool in order to discover patterns using as input *all* the known proteins. Thus, we hoped to compile a *vocabulary of life*, i.e. a set of sequence fragments which are the building blocks used by Nature in the construction of proteins. If such a vocabulary does indeed exist (and there is evidence that it does) then its discovery could offer invaluable insights into the workings of proteins and, possibly, help in the *de novo* engineering of novel, "designer" polypeptides. In Section 3.4 we discuss the results of running the algorithm over the entire non-redundant database (containing all the publicly available proteins) from NCBI, the National Center for Biotechnology Information (<http://www.ncbi.nlm.nih.gov/>). We also address a host of other issues

pertaining to large scale pattern discovery such as appropriate statistical frameworks for deciding which patterns are important and which are within the levels of statistical noise.

Finally, we show how the vocabulary of patterns compiled above can be used in order to perform homology detection searches (in the spirit of BLAST and FASTA) in a novel way. We describe a new tool that we have developed for performing this task and give examples where, using this tool, we are able to annotate previously uncharacterized ORFs. We believe that our approach is a very promising one and that it can prove very helpful in annotating new proteins, especially since current homology-detection tools fail to annotate between 60% and 80% of the newly sequenced genes.

In the final chapter, we discuss some more applications of our algorithm. From the field of Biology we focus on the problem of *multiple sequence alignment* and show how TEIRESIAS can be used as part of a solution to that problem. The second application comes from the area of computer security. In particular, we describe a system (developed in cooperation with the IBM research lab in Zurich) designed to intercept malicious attempts that exploit bugs in popular network utilities (e.g. *ftp* or *mail*). We also present some research directions for the future. Most notable among them is an approach to attack the *protein folding* problem using the vocabulary of patterns compiled from the non-redundant database in conjunction with crystallographic data from the PDB.

1.4 A Biology Primer

This section does **not** intend to be a full fledged introduction to Molecular Biology. There exist excellent books that serve that purpose [114, 70]. Here, we will provide just enough information so that the reader is able to understand the terms used in the thesis.

1.4.1 Introduction

Life is the subject of Biology. What is a valid definition for all things alive? This is not just a question intended to get someone's attention. It is a real concern. To understand why, it helps to realize something that many scientists had difficulty accepting as late as the 1940s. Namely that there is nothing special (a supernatural force, a "vital" quality) that is specific to living organisms. The laws that govern and explain life at the microscopic level are merely the laws of physics and chemistry.

This being the case, it seems natural to ask what distinguishes living from non-living "chemical forms". At what level of complexity, or organization do we draw the line that leads from non-life to life? In the 19th century the answer was thought to be linked to the predominant role of carbon compounds found in living matter (the manifestation of this belief is evident even today in the division of chemistry into "organic" and "inorganic" branches). However, it soon became clear that this is not the case; any carbon-based chemical compound found in living organisms could also be produced at the laboratory. A different answer was needed.

Nowdays, the predominant approach is to characterize life by its *properties*. In particular, a system is thought to be "living" if it has the following three general characteristics:

- metabolism
- growth
- reproduction

The most elementary unit exhibiting these properties is the *cell*. The cell theory, was first suggested in 1839 by the German microscopists Matthias Schleiden and Theodor Schwann. The theory asserts that all living organisms are constructed from small fundamental units, the cells. Furthermore they showed that cells proliferate by the process of *division* during which a cell creates an identical copy of

itself, the so called *daughter cell*. Later on, it was discovered that cells are surrounded by a membrane, the *cell wall* and that the most advanced cells contain an inner body, the *nucleus*, which is surrounded by a *nuclear membrane*.

Since its inception, the cell theory has been put to test numerous times without ever failing: indeed, all living organisms (*unicellular* or *multicellular*) are made up from self-replicating cells ².

By the mid 19th century it was known that higher organisms begin their circle of life as unicellular embryos. Furthermore, it was clear that parents pass on to their children a number of their physical characteristics. It was becoming then obvious that at least the first cell of an embryo must contain a genetic material of sorts that allows heredity information to be transmitted across generations. The branch of *genetics* began as an effort to understand the nature of this material and the precise way in which it is utilized. Great discoveries were made in the process. Their cumulative effect is that today we have the technology to start probing the fascinating machinery of life. Several key facts about cells and heredity are already known. We will look at them in the subsection that follows. After that, a short history of modern genetics is presented. The intention is to put into a historical perspective the achievements of today.

1.4.2 Cells and Heredity

(The discussion in this section is kept as simple as possible. We only present the general concepts behind the phenomena discussed. It should be kept in mind, though, that in reality those phenomena appear with a certain amount of organism-dependent variability. Wherever possible, an effort has been made to focus on only those characteristics of a process that remain invariant across organisms.)

Cells come in two basic varieties. *Prokaryotic* cells are the simplest form, composed of a single compartment which is protected from the outside with a *membrane*

²An interesting variation in the theme of life are the *viruses*. Viruses do not own a cellular mechanism but must infect other cells in order to use the life machinery of the latter.

called the *cell wall*. *Eucaryotic* cells have a central, all encompassing compartment (also protected by a wall) which contains many smaller compartments. The most important among these compartments is the *nucleus*, which contains the genetic material of the eucaryotic cells. Procaryotic cells are found only in bacteria while higher organisms are composed of one or more eucaryotic cells.

Most of the cell (about 90%) is composed of water. The remaining 10% contains two types of molecules:

Elementary molecules: these are small molecules created by a variety of chemical reactions in the cell. Most important among them are the *nucleotides* and the *amino acids*. Elementary molecules provide the building material for polymeric molecules.

Polymeric molecules: these are the structural components of the cell. They are formed when elementary molecules bind together into long chains. The polymeric molecules that we will be focusing on are (i) the *nucleic acids* (DNA, RNA) and (ii) the *proteins*. Nucleic acids are polymeric assemblies of nucleotides while the proteins are chains of amino acid. Nucleic acids and proteins are sometimes collectively referred to as *macromolecules* or *biosequences* or *biological sequences*.

Proteins

Proteins are the most important macromolecules. They are responsible for almost the entire repertoire of biochemical reactions taking place inside the cell. Proteins come in many flavors and with a variety of functionalities. Some of them include:

- *Structural proteins:* They are the building blocks of the various tissues.
- *Enzymes:* they catalyze vital chemical reactions that would otherwise take too long to complete.
- *Transporters:* they carry chemical elements from one part of the organism to another (e.g. hemoglobins carry oxygen).

- *Antibody proteins*: they are part of the immune system.

Because of the wide variety of tasks that they perform, proteins are found dispersed throughout the cell compartments. Several proteins (the so called *receptors*) are partially inside the cell and partially outside the cell. These proteins allow the interaction of the cell with its surrounding environment. In particular, the part that is outside the cell *senses* external conditions and initiates appropriate set of events through the part that is inside the cell (this process is known as *signal transduction*)³.

Proteins are chains of amino acids. Their length can range from just a few tens of amino acids up to several thousand; a typical size is between three and four hundred amino acids. There are 20 different amino acids. All of them have the basic structure depicted in Figure 1.3.(a). More specifically there is a central carbon atom (denoted as C_a and called the *alpha-carbon*) to which four chemical groups attached via covalent bonds: a hydrogen atom (H), an *amino group* (NH_2), a *carboxyl group* (COOH) and a *side chain* (R). What distinguishes the various amino acids is the particular side chain R that each one has. Figure 1.4 shows some amino acids with their corresponding side chains. Table 1.1 lists all the 20 amino acids along with the three- and one-letter codes used for each. The last column of that table classifies the amino acids according to the chemical properties that their side chains bestow upon them. Amino acids belonging to the same class are said to be *chemically similar*.

Proteins are formed by amino acids binding together to form polymeric chains. Binding occurs when the carboxyl group of an amino acid interacts with the amino group of the next amino acid. The result of this process is (i) the formation of a bond between the two amino acids and (ii) the generation of a water molecule from atoms released from the carboxyl group and the amino group (see Figure 1.3.(b)). The bond holding together the two amino acids is called a *peptide bond*. For this reason, a stretch of amino acids is often called a *polypeptide*. Furthermore, the

³Receptor proteins are often the targets of viruses trying to enter the cell. They are also the anchor points that drugs use in order to get hold of particular cells. For these reasons, receptors are especially important in pharmacology.

amino acids that are actually part of a protein are also referred to as *residues*. The reason for the name is the loss of the water molecule during the formation of the peptide bond: the amino acid used in the protein is a “residue” of its free form self.

Figure 1.3.(b) depicts a protein $R_1R_2 \dots R_n$. The repetitive part of a protein (namely, what is left when all the side chains R_i are removed) is called the *backbone* of the protein. Notice also that the amino group of the first amino acid (R_1) and the carboxyl group of the last amino acid (R_n) are not part of any peptide bond. As a result, it is possible to use these two groups in order to impose a notion of direction on a protein. More specifically, a protein “begins” at its *N-terminal point* (i.e. at the amino acid with the free amino group) and ends at its *C-terminal point* (at the amino acid with the free carboxyl group).

DNA

The second macromolecule of interest is the *deoxyribonucleic acid*, better known as DNA. DNA molecules are very long polymeric chains of nucleotides: a single DNA molecule can be millions of *bases* long (nucleotides are alternatively called bases). There are four different nucleotides called Adenine (A), Guanine (G), Cytosine (C) and Thymine (T).

The mechanics of DNA formation are somewhat reminiscent of that of proteins. In particular, nucleotides have a basic part which is common to all of them. What distinguishes one nucleotide from the other is the particular side group that is attached to the basic part. Long chains are formed when series of nucleotides bind together through *sugar-phosphate* bonds. These bonds hold together the basic parts of successive nucleotides, in the same way that peptide bonds hold together the amino acids of a protein (Figure 1.5.(a)). As was the case in proteins, the assembly of the basic parts in a DNA molecule is referred to as the *backbone*. There is, though, one respect in which the structure of DNA is different than the structure of protein: while proteins have only one chain, DNA comprises two interconnected parallel strands of nucleotides (Figure 1.5.(b)). These strands are connected through hydrogen bonds between the side groups of facing nucleotides.

An important point is that there are rules deciding which pairs of nucleotides can face each other in the DNA double strand: in particular, Adenine is always paired with Thymine while Guanine is always paired with Cytocine. Because of these rules, either of the two strands uniquely defines the other and can be used to describe the corresponding DNA molecule. As a result, a DNA molecule can be represented as a single string over the alphabet of the nucleotides. Finally, Figure 1.6 gives a three dimensional picture of the arrangement of the DNA strands in space. This is the famous double helix configuration, proposed by Watson and Crick in 1953.

The entire DNA of an organism comprises that organism's *genome*. The size of a genome depends on the complexity of the host organism. The human genome, for example, has an estimated 3×10^9 *base pairs* (a base pair, or bps, is another way to refer to a pair of facing nucleotides within the DNA double helix). Simpler life forms have genomes whose lengths range from a few thousand bps (viruses) to several million bps (bacteria). In eukaryotes, the DNA is kept inside the *nucleus*. Every cell carries the exact same copy of DNA. Actually, in most eukaryotes the nuclear DNA is not kept as a single molecule but is rather broken up into distinct linear bodies, the *chromosomes*. Chromosomes are used in order to compactly pack the DNA. This is necessary as DNA can be very long. The DNA in a human cell, for example, if fully extended would be three feet long. Clearly some form of coiling is required. Chromosomes achieve this coiling by utilizing a number of proteins that bind the DNA and wrap it around them.

The reason that DNA is so important is that it *is* the genetic material. It carries hereditary information from parents to children. DNA is literally a code, in the computer science sense of the term. What it codes for is proteins. In the same way that a long binary string can be broken into bytes and every byte can be translated into an ASCII character, DNA is broken into triplets and every triplet is translated into an amino acid. The mapping between triplets of DNA bases and amino acids is called the *genetic code*. Since there are $4^3 = 64$ possible triplets of nucleotides but only 20 amino acids, it follows that this mapping is many to one. The genetic code is shown in Figure 1.7. The nucleotide triplets coding for the

amino acids are called *codons*. Notice that there are several codons encoding what is marked as *[end]*. These codons are called *STOP codons* and do not really code for any amino acid; they indicate the end of a protein coding region on a large DNA molecule. Their role will be detailed later.

It is interesting to note that the redundancy present in the genetic code as the result of having several codons coding for the same amino acid, seems to be calculated. It was recently shown [107] that among all possible functions that map triplets of nucleotides to amino acids, the genetic code belongs to the select few with the highest *error tolerance*. In particular, let x be a DNA string and let $GC(x)$ denote the polypeptide that results when the codons of x are translated into amino acids using the genetic code. Assuming that x' results from x with a small amount of *replacements* (i.e. changes of one nucleotide to another), then with high probability $GC(x) = GC(x')$ (x is assumed to be relatively long).

Not all of the genome is used for coding proteins (with the exception of some very primitive organisms with very short genomes). In particular, the protein coding parts of DNA are dispersed throughout the DNA molecule (Figure 1.8). These coding parts are organized in *genes*, i.e. distinct regions of consecutive bases. Every gene codes for one particular protein. Genes are flanked by *control regions* which mark the beginning and the end of these genes. The remaining DNA seems to be non-operational and is known as *junk DNA*. In reality, genes may be themselves interrupted by non-coding regions (Figure 1.8.(b)). This is especially true in higher organisms (viruses and bacteria have usually short DNAs that does not allow the luxury of unused areas). Such regions are called *introns* while the parts that contain useful codons are known as *exons* (for *expressed regions*).

Interestingly enough, the largest part of the genome in higher organisms is composed of junk DNA. In humans for example, about 95% of the DNA is non-coding. This can create some problems when *sequencing* the DNA, as it necessitates the development of methodologies for deciding which DNA regions correspond to genes and which do not. We will return to this issue during the discussion on sequencing.

From Gene to Protein

Genes are passive entities. They just sit there, waiting to be translated into their corresponding proteins. In this idle state, genes are said to be *unexpressed*. Events occurring inside the cell (or originating outside of it) trigger the creation of selected proteins. At that point, the cellular mechanism responsible for the *expression* of genes into proteins takes control. This mechanism locates the appropriate gene(s) on DNA, reads them out codon by codon and synthesizes the corresponding proteins by putting together the amino acids specified by these codons. This process has two steps: (i) *transcription*, which copies the target gene into a DNA-like molecule called *RNA* and (ii) *translation*, which reads the RNA and creates the final protein.

The role of transcription is to make a copy of the part of the DNA that corresponds to the gene of interest. The need to copy the gene has to do with the fact that some proteins *must* be synthesized in particular parts of the cell (the so called *protein localization* problem). It is not possible to generate such proteins in the nucleus and then have them travel to their corresponding destinations.

The material on which the gene copy is recorded is RNA. RNA (ribonucleic acid) is another polymer, also made up of nucleotides. Although RNA and DNA are chemically very similar, there are a few differences. First, RNA uses the nucleotide Uracil (U) where DNA would have used Thymine (T). Second, the backbone-forming basic unit of the RNA nucleotides is slightly different from the corresponding basic unit of the DNA nucleotides (that is why the former are actually called *ribonucleotides*). Finally, RNA is single stranded. The RNA used for gene copying is called *messenger RNA* (or mRNA for short), in order to differentiate it from the RNA used for other cell processes.

Transcription is performed by special enzymes known as *RNA polymerases*. The first thing that an RNA polymerase does is locate the beginning of the gene in question. It does so by recognizing the start region of the gene. In particular, RNA polymerases have the ability to chemically bind the start regions (these regions are also called *promoters* because in some sense they promote the expression of the gene). The promoter provides in that way an anchor point for the RNA polymerase which is now positioned at the right spot to start transcribing the gene. The RNA

polymerase then begins “walking” over the target gene (Figure 1.9). As it does so, it unzips a small part of the DNA helix a few tenths of base pairs in size — the so-called *transcription bubble*). The one of the two single DNA strands thus created is used as a template for the elongation of the progressively growing mRNA molecule. In particular, the RNA polymerase collects free ribonucleotides that abound in the nucleus environment and pairs them with complementary nucleotides of the DNA strand currently read. At the same time, the newly collected ribonucleotides get attached to the already formed mRNA chain. As the RNA polymerase proceeds up the gene, it closes the bubble behind it and opens a new one, continuing the mRNA elongation process. Eventually, the STOP codon of the gene is encountered, at which point the RNA polymerase stops the transcription.

The mRNA molecule thus created is called the *primary transcript*. This molecule contains both the exons and the introns (if any) of the gene just transcribed. Any introns present in the primary transcript are *spliced* out. This is achieved by special proteins that can recognize and remove the introns. The splicing process creates the final version of the mRNA which is called the *functional transcript*. This molecule will then travel to the appropriate location in the cell where it will be translated into the appropriate protein.

The translation phase is responsible for the transformation of mRNA into a protein. This translation is accomplished by the *ribosomes*, complex bodies built from proteins and RNA (the ribosomal RNA is referred to as rRNA). Ribosomes are the chemical factories of the cell. They operate by walking over an mRNA molecule, reading the codons on the mRNA one by one, getting the appropriate amino acid for each codon and binding these amino acids together into the final protein chain (Figure 1.10). In accomplishing this feat, ribosomes utilize *tRNA* molecules.

There is one different tRNA molecule for each possible codon. Each one of them is made of RNA and has the general structure depicted in Figure 1.10.(a). On one end, a tRNA has its *anticodon*. This is a triplet of ribonucleotides. This triplet helps a tRNA molecule bind any mRNA codon X that has a nucleotide triplet complementary to the anti codon of the tRNA (Figure 1.10.(b)). On its

other end, the tRNA molecule has attached the amino acid coded for (according to the genetic code) by the codon X .

Ribosomes have two cavities, each capable of accomodating a single tRNA molecule. tRNAs are placed in these cavities so that their anticodons come in direct contact with the mRNA immediatelly below the cavities. The ribosome marches over an RNA molecule in steps of one codon at a time. At every step it looks at the codon that lies under the rightmost cavity and gets into that cavity a free tRNA (tRNAs exist freely in the cell) whose anticodon is complementary to the codon at hand. The leftmost cavity of a ribosome is occupied by a tRNA used to hold the partially grown peptide chain. As soon as the free tRNA enters the rightmost cavity and binds to the underlying mRNA through its anticodon, an enzymatic process, called *peptidyl transferase*, takes place and attaches the amino acid of the rightmost tRNA to the chain connected to the leftmost tRNA. At that point the chain belongs jointly to both tRNAs. Then the chain is detached from the first tRNA which is discarded, and the second tRNA along with its attached amino acid chain and its underlying codon moves to the leftmost cavity. This movement also forces the entire mRNA to move by one codon, thus revealing the next codon to be translated under the rightmost cavity. The appropriate tRNA then moves in the rightmost cavity and the whole process is repeated. Eventually, the stop codon on the mRNA will signal the end of the translation and the finished protein will be detached from the ribosome.

Figure 1.11 gives a schematic representation of both the trascription and the translation process.

Cell Division

Cell division necessitates copying of the entire DNA of the cell, as this copy must be passed over to the new, daughter cell. The process by which DNA is duplicated is somewhat reminiscent of the trascription phase during the expression of a gene. In particular, the DNA double helix gets gradually unzipped. Each of the original DNA strands acts as a template for the creation of a new, complementary strand (Figure 1.12). As the process proceeds, two new double helices materialize, each

one a copy of the original. One of them will become the genome of the daughter cell.

The new strands are built by appropriate enzymes, the *DNA polymerases*. These enzymes walk along each DNA strand of the parent cell and pair the nucleotides of the strand with the appropriate complementary nucleotides. At the same time they catalyze the generation of the sugar–phosphate bonds between the incoming nucleotides, thus giving rise to the backbone of the new strand. There is a number of other reactions that are part of the duplication process (e.g. DNA must be uncoiled from its highly packed form in the chromosomes) which we are not going to discuss here.

Mutations

The DNA of all organisms suffers a number of *mutations*. Such mutations can occur at level of single nucleotides or at the level of entire chromosomes.

Macroscopic mutation events affect entire genes, sometimes even chromosomes. They can occur during the DNA duplication process. *Translocations*, for example, occur when DNA is interchanged between chromosomes. Another type of mutation occurs when a part of DNA flips its orientation relative to the chromosome it belongs to. Such problems arise because of the mechanics of the DNA replication process. Because of the level at which they happen, macroscopic mutations are usually called *chromosome rearrangements*. The results of such wide ranging mutations can be devastating: entire genes can be lost or become non–functional. In such cases, the result is that the new cell that comes out of the cell division process dies shortly after it is created.

A second type of mutation (and the one that this thesis mostly focus on) involves changes of only one or just a few bases. Such mutations can be categorized as follows:

Replacements which occur when one DNA base is changed into another. Mutations of this type are also called *point mutations*.

Insertions can introduce one or more nucleotides between any pair of successive DNA bases.

Deletions remove consecutive bases from a DNA molecule.

The mutations described above can happen either during the replication of DNA because of mistakes made by DNA polymerases, or, as a result of normal cellular operations which sometimes create such mutations as a side effect. The mutations in this latter case are called *spontaneous*. The rate at which spontaneous mutations occur is organism-dependent and is referred to as the *background level*. Several events (e.g. presence of pathogenic compounds in the cell, malfunctioning proteins, radiation) can increase the mutation rate above the background level.

Spontaneous mutations tend to happen randomly over the genome, both in coding and non-coding regions. Those in non-coding regions are not usually a problem. Mutations on genes, however, have a direct consequence on the proteins coded by those genes. For example, point mutations can:

- create a stop codon where no one previously existed. This shortens the resulting protein, possibly removing regions that are critical for the operation of the protein.
- change the amino acid encoded by the codon affected by the mutation. Depending on the importance of the encoded amino acid the impact on the encoded protein can range from mild (the protein containing the amino acid can still perform its task) to grave (critical functionality is lost).

It is also possible that a point mutation on a gene does not change the corresponding protein at all. This happens if the changed codon still codes for the same amino acid. Situations like this arise because of the redundancy built into the genetic code. Mutations of this type are called *silent mutations*.

Finally, insertions or deletions of nucleotide strings whose length is not a multiple of three result in *frame shifts*, i.e. in changes to the delineation of codons. Again, the mutated gene might code for a non-functional protein.

It is interesting to note that mutations, despite their potentially devastating results, are the mechanism that Nature uses in the evolution of the species. As the environment constantly changes it becomes imperative for the various organisms to

change as well or they become obsolete. Mutations are the agents of this change. While they may lead to malfunctioning proteins (which are not favored by the evolution i.e. they die) they also allow the generation of new, mutant genes. Such genes can often be beneficial for the adaptation of an organism into a changing environment.

Mutation Matrices

Molecular biologists learn a lot about a new protein just by comparing its sequence with the sequences of other already known proteins. The most popular method to compare two protein sequences involves their optimal (or near-optimal) alignment through the use of biologically relevant edit operations (replacements, insertions, deletions). Such alignments are scored by aggregating the *costs* of the particular edit operations that were involved.

The use of appropriate costs for weighting the various edit operations is an extremely important issue in biological alignments [1, 2]. Their importance is more pronounced when comparing proteins which are evolutionary distant (i.e. their respective sequences are quite different). In such cases, selection of the proper costs can be the deciding factor in recognizing the biological relation of the proteins or not. The determination of these costs is achieved with the combination of biological knowledge, statistics and the meticulous study of existing data in order to uncover hidden evolutionary relationships.

A description of the exact techniques used in order to compute biologically sound costs is outside the scope of this document (details can be found in [41, 25, 45]). However, we will be using throughout this thesis the outcome of these techniques, namely the resulting *mutation matrices* (also known as *amino acid substitution matrices*). Mutation matrices give the probabilities of one amino acid mutations. A mutation matrix is a square, 20×20 table with real or integer entries where every row and every column correspond to some amino acid. In particular, the (i, j) -th entry of such a table contains a score which describes the probability of the i -th amino acid mutating to the j -th amino acid (the amino acid ordering is arbitrary). This score is usually a normalized logarithm of the real probability;

logarithms are used in order to employ additions rather than multiplications (which would be the case if actual probabilities were used) in the computation of aggregate scores due to successive edit operations.

The most widely used families of substitution matrices are the PAM [25] and BLOSUM [45] matrices. Each family contains a number of members; depending on how distant the compared sequences are, a different member–matrix might be needed in order to “correctly” score an alignment.

DNA Sequencing

From the early seventies and on, a number of discoveries (known today collectively as *recombinant DNA technology*) have made possible the manipulation of the cell’s DNA in a variety of ways. For example, it is now possible to cleave DNA molecules at specific cut–points, insert DNA from one organism into another, have one organism expressing genes that belong to another organism and, lately, the ability to clone mammals!

Recombinant DNA technology is based almost entirely in exploiting existing cellular mechanisms. The cells themselves need to manipulate DNA in complicated ways (recall the transcription and translation mechanisms). As a result, they have developed specialized enzymes that operate on DNA in a multitude of modes. Molecular biologists identify and isolate such enzymes, and harness them in order to handle genetic sequences in specific ways. One such example is provided by the technique used for deciphering the sequence of a protein. It is easy to isolate arbitrary mRNA molecules as they travel towards the ribosomes that will translate them. Remember that these molecules are the result of the functional transcript of a gene, i.e. any existing introns have been removed. This is a wonderful situation because nature has already taken care of one of the problems associated with sequencing, namely splicing out introns. So, it seems that in order to identify the amino acid sequence of the relevant protein all that is needed is to sequence the mRNA molecule, i.e. find the ribonucleotides that it is made of. After that, the problem reduces to translating codons to amino acids, a trivial task since the genetic code is known.

However, although existing techniques make it possible to sequence DNA, there are technical reasons that make it very hard to use these techniques on mRNA. Fortunately, nature has provided a shortcut. *Retroviruses* are viruses that carry their genetic information in RNA rather than in DNA. When invading a cell, a retrovirus needs to transform its RNA into DNA so that it can be implanted into the genome of the attacked organism. For that reason, retroviruses possess an enzyme called *reverse transcriptase*. This enzyme performs the reverse job from an RNA polymerase: it walks along an RNA strand and builds a complementary DNA strand. Such enzymes can help transform an mRNA molecule into an “equivalent” DNA double helix in the following way: first, the reverse transcriptase generates a complementary DNA strand for the mRNA at hand. Then the single DNA strand is processed by a DNA polymerase (the enzyme used to duplicate the DNA during cell division) which generates a full fledged DNA double helix out of it. The DNA molecule thus produced is called *complementary DNA* or *cDNA* for short. With the cDNA available, we are almost done: all that is left is to apply standard DNA sequencing techniques on it.

In the late 80's it was becoming apparent that several of the recombinant DNA techniques could be combined in order to allow the sequencing of long DNA molecules. These techniques became the enabling technology behind the Human Genome Project. As mentioned at the beginning of the chapter, the objective of this project is the sequencing of the entire genome of selected organisms (humans, among them).

There is a number of technical challenges associated with this task. The most salient among them is the sheer length of a complete genome. For higher organisms it is in the order of billions of bps while even for bacteria it can reach several millions base pairs (the genome of *E. Coli*, one of the first bacteria to be sequenced, is about 5 million bps). Put simply, the problem is that we cannot take a long DNA molecule, uncoil it, put it under some microscope and read its bases one by one. As a result, other indirect methods must be employed in order to infer the nucleotides in a DNA helix.

Such methods have indeed been devised. *Gel electrophoresis* is probably the

most well known among them. Even these methods, though, have their limitations. In particular, they can be used to sequence only medium size molecules, typically of no more than a few thousand bps. Overcoming this restriction involves (i) breaking up DNA in small parts of a few thousand bps (this process is called *digestion*), (ii) sequencing these small parts individually, and (iii) putting them back in the appropriate order so as to restore the original DNA molecule.

The difficulty here is in the last step: when a long DNA molecule is broken, the resulting segments lose their order information. To address this problem, a long DNA molecule must be treated many times: each time the molecule (actually, a clone of the original) is digested in a different way. Having sequenced the smaller DNA pieces from each digestion, the next step is to build a *tiling path* that will hopefully allow one to reconstruct the original molecule (Figure 1.13). In order to make sure, though, that there exist enough overlapping segments to allow the reconstruction of the original molecule, many different digestions are needed and this increases the cost of sequencing. Furthermore, from a theoretical point, the reconstruction problem may not have a unique solution as more than one assemblies might be possible.

The bottom line is that producing *physical maps* of genomes, i.e. resolving DNA sequences at the level of individual nucleotides is a complicated and time consuming process. Fortunately, there are also techniques that allow the creation of alternative descriptions of the genome. A *genetic map* is a linear representation of a chromosome where different genes are represented as linear segments and the relative distance between successive segments is proportional to their distance (in bps) on the real chromosome. In order to construct a genetic map one does not have to actually sequence the genes involved. Other studies (called *gene linkage studies*) can be used. Figure 1.14 gives an example of a genetic map.

Finally, another issue that arises in sequencing genomes with a lot of junk DNA (as is the case in humans) is the following: how can we know if a DNA region just sequenced is non-coding, coding, or a combination of both (which can be the case if it corresponds to part of a gene that has both exons and introns)? Determining if this region contains an entire gene (or part of it) is relatively easy if cDNA for

this gene is available: capturing functional mRNA transcripts and turning them into cDNA is a process that proceeds independently of the genome sequencing. As a result, data bases of cDNA have been developed. When a new DNA region s becomes sequenced, it is compared against such data bases. If s does indeed contain a gene part, then the general form of s will be as follows:

$$s = XE_1I_1E_2I_2 \dots E_nI_nY$$

where E_i correspond to exons, I_i to introns and X, Y to other non-coding regions or parts of other genes. If a cDNA database contains a cDNA that has $E_1E_2 \dots E_n$ as a substring, then we know that s contains a gene. Chance matches are not a problem here since the exons E_i are relatively long (several tenths of nucleotides at least).

Otherwise one of two things happen: either s is a non-coding region or it contains (part of) a gene for which no cDNA has been captured. In order to decide

which of the two is true, methods have been devised which exploit the fact that in all organisms the nucleotide composition of coding regions is different than that of non-coding regions. Genome regions which are deemed as coding under these methods are referred to as *open reading frames* or *ORFs*. This terminology indicates that they probably mark the beginning of a gene.

Protein Structure

Up to now, we have been thinking about proteins as sequences of amino acids. This will be the norm throughout this dissertation. The representation of a protein as a string of its constituent amino acids (always starting at the N-terminal of the protein) is called the *1-dimensional* representation of the protein or, more often, its *primary structure*.

It is, however, important to remember that proteins are molecules. As such, one of their most defining characteristics is their *fold* i.e. the conformation that they assume in space. The fold of a protein is of extreme importance because it typically provides clues about the function of the protein.

The ability of a protein to fold is due to the flexibility of the covalent bonds of its backbone. Figure 1.15 makes this point graphically. First of all, the nature of the bonds is such that the nitrogen and carbon atoms involved in every peptide bond, along with their accompanying hydrogen and oxygen atoms, are coplanar. This is indicated in the figure by a plane that contains them. This plane can rotate around the covalent bonds that connect it with the two adjacent *alpha*-carbon atoms. The angles of rotation are denoted as ϕ_i and ψ_i . The possible combinations of angles are dependent on the side groups of the protein which impose steric restrictions. The forces acting among the atoms of the protein chain (being hydrogen bonds, ionic bonds, hydrophobic interactions or Van der Waals attractions) decide the particular values for the ϕ_i and ψ_i angles.

There is clearly the possibility for infinite variability in the structure of different proteins. But, it turns out that there is a small number of basic local structures (each a few tens of amino acids) that are used again and again by proteins. Every protein is built from a number of copies of these basic building blocks, each protein combining the blocks in its own, idiosyncratic way. Figure 1.16 describes one such basic local structure, known as the *α -helix*. As the name implies, this structure has a helical conformation. The presence of hydrogen bonds between hydrogen atoms of the amino group and oxygen atoms attached to the carboxyl carbons give the helix its spring-like form. The helix completes a full turn every 3.6 amino acids. The length of naturally occurring helices ranges from below ten amino acids to several dozen of them.

The 3-dimensional structure of a protein can be described at several levels. The *secondary structure* focuses on basic building blocks like the *α -helix*. At this level of detail, one is concerned with the identification of relatively short regions of a protein (a few amino acids) and their characterization in terms of a few well known basic structures (*α -helices*, *β -sheets*, *β -strands*, *helix-turn-helix* combinations etc.). Basic structures at that level are also known as *motifs*. There is also the *tertiary structure* of a protein: here, several consecutive motifs are grouped together forming *domains*. Domains are potentially long regions of proteins (can be in the hundreds of amino acids) and represent functional units: every domain per-

forms an individual role within the overall protein function. Consider for example the RNA polymerase protein mentioned during the discussion of the transcription of DNA into mRNA. We mentioned before that the polymerase first needs to bind the start region at the beginning of the gene. Recognizing and binding this start region is a distinct part of the protein's activity. The part of the protein that is responsible for this task constitutes a domain. The importance of domains stems from the fact that a particular domain may appear in many different proteins. Furthermore, every time it appears it always performs the same task. In summary, domains are structural blocks which are delineated based on the function that the block performs.

Figure 1.17 shows both the primary and the tertiary structure of a hemoglobin protein (responsible for the transfer of oxygen) from the organism sea cucumber . The tertiary structure is made up of six α -helices. Each helix is a secondary structure element. The amino acid regions of the protein which code for the helices are surrounded by boxes on the primary structure of the protein sequence. Notice that the tertiary representation describes basically the backbone of the protein and the various secondary structure elements are represented in a cartoon-like way. This is a standard representation and is used because it makes it easy to depict graphically the structure of the protein. *Ball and stick* models can also be used but the resulting picture is messy.

In conclusion, it can be said that knowing the three dimensional structure of a protein is the ultimate solution in understanding the function of this protein. Unfortunately, existing techniques for solving protein structures, such as X-ray crystallography and NMR, are very cumbersome and time consuming. Furthermore, there are many proteins which (for physical reasons) are not amenable to study by either of these two techniques. It is not then surprising that a lot of research is focused in devising computational methods for *predicting* the structure of a protein from its sequence of amino acids. This is the famous *protein folding problem*, decidedly the most important unsolved problem in Molecular Biology today. A key property of proteins (and the one that make people believe that computational techniques are an option) is that proteins always assume the same conformation.

So, although there is an infinite number of folds that a protein can assume it always selects the same one. According to physicists, this is the conformation that minimizes the protein's energy. This “unique fold” feature indicates that all the information needed for a protein to fold is in its amino acid sequence.

1.4.3 History

Biology and especially Genetics have a fascinating history. Here, we will just highlight a few of the most important facts. The intention is to put the research performed today in Molecular Biology into perspective. The story begins at the mid 19th century. To appreciate the events, recall that the scientific environment at that time stood as follows: the cell theory had just been proposed, people were speculating that organisms start their life as unicellular embryos and it could be observed that children bear resemblance to their parents. It could thus be inferred that there was some form of a genetic material on which parental features could be passed down to their offsprings. Clearly, this material must be hiding somewhere in the cell. But what is it exactly? And how is heredity information recorder and used? These were the questions that the biologists of that time, with their very rudimentary technology were trying to answer.

Biology studies life. It seeks to define what “life” is, identify the laws governing it and explain how these laws are related to the observable expressions of life. Different branches of Biology address these questions at different hierarchical levels. Historically, the first efforts to study living organisms were at the *macroscopic* level, by focusing on organisms, populations, communities and ecosystems. This was in part unavoidable since technology that would allow examination of life at the *microscopic* level did not exist. Important discoveries were made nonetheless. The most salient among them was the work of Charles Darwin. In the second half of the 19th century Darwin, comparing the structural characteristics of many different species, was led to propose his famous *theory of evolution*, arguably the most passionately debated postulate in the entire history of science. The center stone of this theory is the concept of *evolution* or *diversification* which asserts that all life has evolved in a tree-like manner where branching points indicate

diversification events leading in the generation of new species from existing ones. Species can be placed in different nodes of this *evolutionary tree* based on the amount of similarity they exhibit.

Around the same time, Gregor Mendel was discovering the basic facts of heredity. A monk in Austria and an amateur biologist, Mendel experimented with the cultivation and cross-breeding of peas. What he was studying was a number of external characteristics like color (e.g. yellow or green) and shape (e.g. round or wrinkled) of the peas. By using plants that had different combinations of such characteristics and by hybridizing them, Mendel was able to deduce that each of the plant features he examined seemed to be controlled by a *distinct* heredity unit. Furthermore, these units (which would be later called genes) seemed to operate independent of each other and followed simple combinatorial rules. For example, when he cross-bred two types of peas that were identical in every respect except for their color (one parent was color A while the other was color B), he found out that *all* the resulting plants (the children) were of color A . When these children were combined among them now, the resulting second generation was 75% color A and 25% color B . This made Mendel realize that the color B *phenotype* (the term “phenotype” denotes the particular value of a feature like color in an individual; so we say that a person has a “blue eyes” phenotype or a “red hair” phenotype) was not lost in the first generation but rather remained dormant. To explain the re-emergence of this phenotype in the second generation, Mendel proposed the following model. The color gene has several variations, each variation defining a single color (today such gene variant are called *alleles*). Let the variation that codes for color A be G_A and let G_B be the allele for the color B . Mendel further postulated that every plant carries color genes in pairs. In the original, purely bred parents the gene pair is the same: the *genotype* of one parent is (G_A, G_A) and of the other parent is (G_B, G_B) (the “genotype” of an individual describes the particular gene variants the individual has). When two peas are cross-bred, the children take one gene from each parent. As a result the first generation offsprings all have the genotype (G_A, G_B) . Since, however, all these offsprings have the color A phenotype, Mendel concluded that the two color alleles are not equivalent in

power: in particular, G_A is the *dominant* allele while G_B is the *recessive* allele and the dominant allele decides the value of the characteristic associated with the gene (in Mendel's case, the characteristic was color). This explanation describes why the color B phenotype is dormant in the first generation of hybrid peas. Looking now at the parents of the second generation, they all have the (G_A, G_B) genotype. As a result, the children of such parents are expected to have (according to the heredity mechanism described above) either of the following three phenotypes:

$$(G_A, G_A), (G_A, G_B), (G_B, G_B).$$

The particular genotype of a child would depend on which color allele it inherits from each parent. Assuming that a parent will pass either allele with equal probability, one expects 25% of the (G_A, G_A) genotype, 25% of the (G_B, G_B) genotype and 50% of the (G_A, G_B) genotype. Keeping in mind that G_A is the dominant allele, this means that one expects to have in the second generation of peas 75% of the color A phenotype and 25% of the color B phenotype, i.e. exactly what Mendel observed in his experiments. The above heredity model together with the dominant/recessive allele hypothesis fit the experimental data perfectly.

The main contribution of Mendel's work was the definition of the concept of the gene (although "gene" was not the word that he was using): one *distinct* entity controlling the value of one characteristic. This was a pretty close approximation to the model that we have today, i.e. one gene controls one protein. Mendel was very lucky in that he chose to study characteristics that happened to be controlled by a single protein. This is the exception rather than the norm: it is now known that most macroscopic characteristics (e.g. the eye color) are arbitrated by *several* proteins. Had he chosen one of these, it would have not been that easy to arrive at the gene model that he proposed. Actually, several people before him tried similar experiments and failed to get results exactly because they chose to study *multigenic* features. Mendel was lucky in other ways too. For example, not all alleles are related through a dominant/recessive relationship. Some of them are *codominant*: in fruit flies, children of one red-eye parent and one white-eye parent have pink eyes.

Despite all the coincidences, the fact is that Mendel guessed the basic properties of genes quite accurately. He published his results in 1865. His work, however, went unacknowledged despite his efforts to promote it. His heredity model, although explaining observed the plant phenotypes, was thought to be too arbitrary: it used too many assumptions (genes are carried in pairs, alleles, dominance/recessiveness, etc.) that in 1865 were difficult to believe. Furthermore, the failure of other people that did similar experiments to get similar results made Mendel's work seem like an accident rather than the bull's eye hit that it really was.

Mendel's theory would remain dormant until new information would corroborate his far reaching assumptions. This information came in the form of new discoveries about what happens inside the cell. First, in 1868 Ernst Haeckel observed that sperm cells are mostly made from nucleic material. Since it was known that sperm and egg cells come together in forming the first embryonic cell, he postulated that the elusive genetic material resides in the nucleus of cells. Then, it was discovered that the nucleus of eucaryotic cells is populated by a number of linear organelles (the chromosomes). These linear bodies exhibited a very distinctive behavior during the cell division process: they were duplicated and divided between the two resulting cells, unlike other cell particles that stayed with the original cell. Furthermore, it was found that sperm and eggs cells (the *gametes*) have in their nucleus half the chromosomes that the other regular cells have. Finally, using microscopy it was observed that the chromosomes of eucaryotic cells could be grouped together into morphologically similar pairs.

So, by the end of the 19th century all the information was really there. All that remained was to make the connection with Mendel's forgotten results. This missing piece was contributed by Walter Sutton in 1903 [102]: he observed the relationship between the chromosome duplicity in cells and the gene duplicity proposed by Mendel. Sutton suggested that Mendel's genes live on chromosomes. He supported his claim by the fact that the gametes get only one chromosome from every homologous pair. Sutton correctly guessed that this mechanism was implementing Mendel's hypothesis that children form gene pairs by getting one gene from each parent.

Sutton's work was extremely influential as it reinforced Mendel's gene model in the consciousness of Sutton's contemporaries. Research now had a model to focus on. A first question had to do with the alleles. How do they occur? One hypothesis was that alleles arise by mutations accumulated on genes across generations.

The legendary American biologist Thomas Morgan set out to check this hypothesis. He realized that he could not use peas like Mendel did: he needed an organism that (i) can be bred in laboratory conditions, and most important (ii) produces new generations fairly often. He selected *Drosophila melanogaster*⁴, a very small fruit fly variety that takes 14 days to produce a generation. Breeding experiments with populations of this fly started in 1908. Morgan and his group hoped to witness the production of a new observable feature. And eventually it happened: after several generations they observed that one of the flies had white eyes, a phenotype that had not been observed before in free living flies. A new eye-color allele had been artificially created, substantiating the claim that genes mutate naturally over time. But it was more than just that: in fact a new species had been created, as there were no naturally occurring white eyed fruit flies. Morgan's proof of the gene mutation hypothesis seemed to unify Mendel's and Darwin's theories!

At the same time with Morgan's work other biologists and biochemists were focusing on revealing the chemical nature of the genetic material. Sutton's work suggested in a very substantiated way that this material was hiding in the chromosomes. But what was it exactly? What was its chemical make up? How were genes positioned on it? By Sutton's time it was already known that chromosomes are mostly made of protein and nucleic acid. So, there was speculation that either of the two should be the genetic material. Actually, this speculation was kind of lopsided: most biologists believed that proteins carried heredity. This belief was not unfounded: proteins were already known to be of grave biological importance; all known enzymes and practically all substances that have been found to play an active role within cells were proteins. On the other hand, DNA had no apparent function. As a result, most people believed that DNA had some secondary role,

⁴Incidentally, *Drosophila* is among the organisms whose genome has been chosen for sequencing by the HGP.

providing merely a substrate for the proteins to operate. The outcome of this misconception was that most of the research regarding the chemical structure of the genetic material was actually focused on the wrong direction: proving that proteins carry the heredity information.

Given the environment described above, it is not surprising that the work of Avery, MacLeod and McCarty in 1944 [7] which implied that the genetic material is really the DNA, created a sensation. Their work was based on previous results by Frederick Griffith in 1928. In his effort to find a vaccine for pneumonia, Griffith experimented with two strains of a pneumonia-causing bacterium. One of the strains (called the S type) was virulent while the other (the R type) was not: inserting S type bacteria into mice would kill them, while R type bacteria had no effect at all. The contribution of Griffith was the following observation: when mixing heat-killed type S bacteria with type R bacteria the result was the transformation of the benign R type bacteria into virulent S type bacteria. Furthermore, the change was of a hereditary nature, as the transformed bacteria gave rise to new virulent offsprings. Based on these observations, Griffith hypothesized that heat left the genetic material of the virulent strains unchanged and this material could leave the dead cell, find its way into the R type bacteria and somehow insert itself into the recipient cells' genetic machinery. Other researchers following Griffith managed to actually isolate and purify the transformation inducing extract of the heat-killed S bacteria.

Avery and his co-workers, managed to identify the extract's nature as nucleic acid. The approach that they followed is in spirit very similar to the one described in the discussion of turning mRNA to cDNA. More specifically, they used three types of enzymes: the first (a recently obtained pancreatic deoxyribonuclease) was capable of breaking DNA into its constituent nucleotides (a process known as *degradation*) while leaving both RNA and protein unaltered; the other two enzymes had similar degrading activity but for RNA and proteins respectively. The experiment that led to the identification of DNA as the genetic material was the following: when the transforming extract from the S type bacteria was treated with the pancreatic deoxyribonuclease before being mixed with the R type bacteria it

would lose its transforming power. This, however, was not the case when treated with the other enzymes: in those cases the transforming power persisted.

Further work by Hershey and Chase [47] in 1952 removed any doubt that DNA was indeed the genetic material. For their experiments they used T2, a bacterium attacking virus. By 1952 it was known that viruses comprise almost entirely of protein and DNA and that they operate by implanting their genetic material into the cell under attack. Hershey and Chase isolated T2 viruses and labeled their protein and DNA with different radioactive isotopes. These marked viruses were then used to infect bacteria. When the infected bacteria were radioactively scanned, they were found to contain only the isotope that was used to mark the viral DNA. The conclusion was that in viruses, as with the pneumonia bacteria of Griffith, DNA is the agent of heredity.

The final challenge was that of solving the structure of DNA. Since Sutton's time it was already understood that the genetic material must have a structure that enables its duplication during cell division. In 1949, Erwin Chagaff used paper chromatographic techniques to discover that the number of adenine (A) nucleotides in various DNA samples was exactly the same as the number of Thymine (T) nucleotides and the same was true for the pair Guanine (G) and Cytosine (C). Then, in 1952, Wilkins and Franklin were able to obtain high quality pictures of the DNA molecule by using X-ray diffraction methods. Their pictures revealed that DNA had a helical structure and that the molecule was composed by more than one nucleotide strands. Around the same time Francis Crick and several others had just finished a theory analyzing the diffraction of X-rays caused by helical molecules (this work had actually originated from an effort to analyze the diffraction data obtained from α -helices in proteins). Using this theory Crick and Watson [113] proposed the double helix model for DNA (Figure 1.6). They did so by using Crick's theory in order to find the stereochemically most stable configuration fitting the data of Wilkins and Franklin.

1.5 Biological Data Repositories

Right from the beginning of the Human Genome Project it was very clear that the success of the entire effort would be crucially dependent on the dissemination of the collected data; the issue of organizing the data in an easily accessible way was a key consideration from early on.

Today there are several public data bases that provide users with the ability to access and work with the available information. This information includes coding and non-coding DNA regions for various organisms, genetic maps, protein sequences, three-dimensional structures, family signatures etc. More important, these data repositories are interconnected and allow users to navigate the available information in smart ways. It is not uncommon today to chemically characterize a new protein (completely or partially) based on the properties of already known proteins similar to the new one. In the words of Stephen Oliver [115]:

“In a short time it will be hard to realize how we managed without the sequence data. Biology will never be the same again.”

Describing the practices of modern day biologists and bioinformaticists is out of the scope of this work (for those interested in the subject, [10] provides a good exposition). Throughout this document, however, we use data obtained from existing data bases. Before proceeding any further then, it is helpful to have some general idea about what these data bases are.

The simplest data repositories available are *sequence* data bases. They contain sequence data for proteins. An entry usually contains the actual sequence of amino acids (or nucleotides of the relevant gene), the organism to which the protein belongs, its function (if known) as well as other information (e.g. relevant publications, the person/laboratory that submitted the sequence etc.). There are three main data bases of sequence data which have been at the center of the HGP effort from the very beginning: GenBank in the United States, the EMBL data base in England and the DNA data base of Japan. All three of them contain *genomic* data, i.e. nucleotide sequences from a multitude of organisms, including humans. Individuals and laboratories can submit sequenced DNA regions using

appropriate procedures and forms. The three data bases form a consortium with the objective to collect and distribute publicly available sequences. Towards this end, they synchronize on a daily basis by exchanging data. Since, however, data come from many sources there is always the possibility that a given sequence has been entered more than once.

It is generally undesirable to have multiple copies of a sequence within a data base [2]. For this reason the National Center for Biotechnology Information has automated a process for processing the aggregate contents of the three data bases mentioned above and producing a *non-redundant* data base with only one copy per sequence.

Another widely used sequence data base, and the one that we will mostly refer to in this work, is SwissProt [8]. It contains amino acid sequences that have been obtained from reverse transcription of mRNA (this procedure was described in Section 1.4.2). What makes SwissProt particularly attractive is the fact that it is *curated*: every protein entry has been checked for sequencing errors and it is accompanied by a description of the protein (function, known domains), references to relevant publications and other information. The availability and reliability of this information is a major asset in the type of correlation inferences that are habitually made when comparing sequences with similar primary structures.

Sequence data bases are, in a sense, the most rudimentary tools for representing genomic information: they just make the plain data available. A lot of additional information, though, can be extracted by processing (in biologically sensible ways) that data. Several data bases offer the results of this kind of processing. The PROSITE [9] repository contains descriptors of protein families. Descriptors are constructed by first grouping together proteins which are *known* to be biologically related and then by identifying (manually!) commonalities shared by these proteins. The commonalities are represented either as regular expressions or as *profiles* (Figure 1.18). A typical PROSITE entry contains a set of related proteins along with their signature (i.e. their corresponding pattern or profile). The intention (not always attainable) is for the signature to be *diagnostic* of the family, i.e. to be able to capture all the family members while disregarding all sequences

that do not belong to the family under consideration. PROSITE is a very useful resource because it incorporates biological expertise: the grouping of the proteins into families is based on existing biological knowledge and is not (as is the case with other systems, like the PRODOM data base to be discussed shortly) based on automatic processing. PROSITE uses SwissProt as the underlying data base from which the proteins are drawn.

A similar system is BLOCKS [44, 46]. It also provides descriptors for families of proteins, the descriptors in this case being profiles. Unlike PROSITE, the profiles used by the BLOCKS data base are constructed automatically. Also unlike PROSITE, BLOCKS allows a family to have more than one descriptors. These descriptors (called *blocks*) are actually constructed by processing the PROSITE families. Every family gets assigned an ordered collection of blocks with the properties that (i) at least half of the proteins in the family contain all the blocks, (ii) in all these proteins the blocks always appear in the same order, and (iii) all block appearances are not overlapping. This collection of blocks becomes the descriptor of the protein family. A more detailed discussion of how the BLOCKS data base is built is given in Section 4.2.3.

PRODOM [24] (the PROtein DOMain data base) is yet another repository of descriptors built on the proteins of SwissProt. PRODOM produces a clustering of SwissProt by first finding homologous regions shared by many proteins and then placing all these proteins together in a cluster. The alignment of the sequences implied by their homologous regions (see again Figure 1.18) gives rise to a profile that becomes the descriptor for the family. The profiles of PRODOM are quite lengthy (several tenths of amino acids) and are usually descriptors of domains.

Finally, there exist *structural* data bases that contain information about the structure of proteins. The most well known among them is the Protein Data Bank (PDB) maintained by the Brookhaven National Laboratory which contains protein structures obtained from crystallographic and NMR studies. Every entry of the PDB data base describes the structure of one specific protein. An entry begins with a title section which gives general information like the protein name, the publication where the work was described, the authors, the organism where the

protein was taken from etc. The title section is followed by the primary structure of the protein, i.e. the sequence of the amino acids it is made of. After that, a section describing the secondary structure of the protein is provided. In this section regions of the protein are characterized in terms of the basic secondary structure elements (α -helices, β -sheets, loops etc.). Finally, the actual three-dimensional structure of the protein is described by giving the coordinates of the protein atoms. More details for the entry format as well as examples can be found in the web site <http://pdb.pdb.bnl.gov/>.

1.6 Research Portfolio

The work upon which this thesis is based can be found in the following documents:

Papers

- “Combinatorial Pattern Discovery in Biological Sequences: The TEIRESIAS Algorithm”, joint work with I. Rigoutsos, *Bioinformatics*, 14(1):55-67, 1998.
- “Motif Discovery in Biological Sequences Without Alignment or Enumeration”, joint work with I. Rigoutsos, *Proceedings of the Second Annual International Conference on Computational Biology*, pp. 221-227, 1998
- “MUSCA: An Algorithm for Constrained Alignment of Multiple Data Sequences”, joint work with L. Parida and I. Rigoutsos, *Discrete and Applied Mathematics*, accepted for publication, December 1998.
- “Sequence Homology Detection Through Large-Scale Pattern Discovery”, joint work with I. Rigoutsos and L. Parida and G. Stolovitzky and Y. Gao, accepted for publication, *Third Annual International Conference on Computational Biology*, April 1999.
- “Unsupervised Hierarchical Motif Discovery in the Sequence Space of Natural Proteins”, joint work with I. Rigoutsos and C. Ouzounis and L. Parida and G. Stolovitzky and Y. Gao, *Proteins*, submitted, *Proteins*, 1998

IBM Technical Reports

- “Unsupervised Hierarchical Motif Discovery In the Sequence Space of Natural Proteins” joint work with I. Rigoutsos, C. Ouzounis, L. Parida, G. Stolovitzky, Y. Gao
- “A Tool for Discovering Sequence Similarity Using Patterns” joint work with I. Rigoutsos, L. Parida, G. Stolovitzky, Y. Gao
- “On the Time Complexity of the TEIRESIAS Algorithm”, joint work with I. Rigoutsos
- “Combinatorial Motif Discovery In Biological Sequences Using the Teiresias Algorithm” joint work with I. Rigoutsos.
- “Tandem-repeat Detection Using Pattern Discovery with Applications in the Identification of Yeast Sattelites”, joint work with G. Stolovitzky, I. Rigoutsos, L. Parida, Y. Gao

Patent Applications

- “Method and Apparatus for Pattern Discovery in Protein Sequences”, joint work with I. Rigoutsos
- “Method and Apparatus for the Data Compression Utilizing Efficient Pattern Discovery”, joint work with I. Rigoutsos
- “Method and Apparatus for Pattern Discovery in 1-Dimensional Event Streams”, joint work with I. Rigoutsos
- “Method and Apparatus for Discovery, Clustering and Classification of Patterns in 1-Dimensional Event Streams”, joint work with I. Rigoutsos
- “Method and Apparatus for Intrusion Detection In Computers And Computer Networks”, joint work with M. Dacier, H. Debar, I. Rigoutsos and A. Wespi.

- “MUSCA: An Algorithm for Constrained Alignment of Multiple Data Sequences”, joint work with L. Parida and I. Rigoutsos

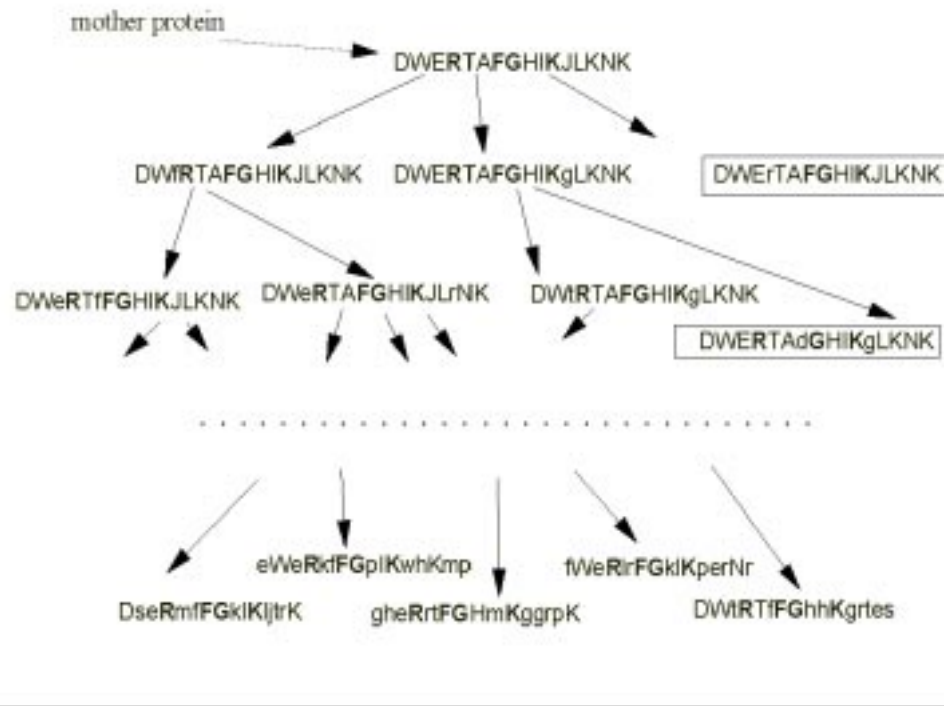


Figure 1.2: The evolution over time of a primordial protein (the “mother protein” occupying the root of the tree), through a series of mutations, into its present-day instances (the leaves of the tree). Mutated amino acids are indicated by lower case letters. The amino acids which are vital for the operation of the protein are marked in bold face. Mutations that affect these amino acids are shown boxed, indicating that the resulting proteins do not survive. Notice that the vital region of the protein can be modeled by the pattern

R..FG..K

where the ‘.’ (the *don't care* characters) represent positions which can be occupied by arbitrary amino acids.

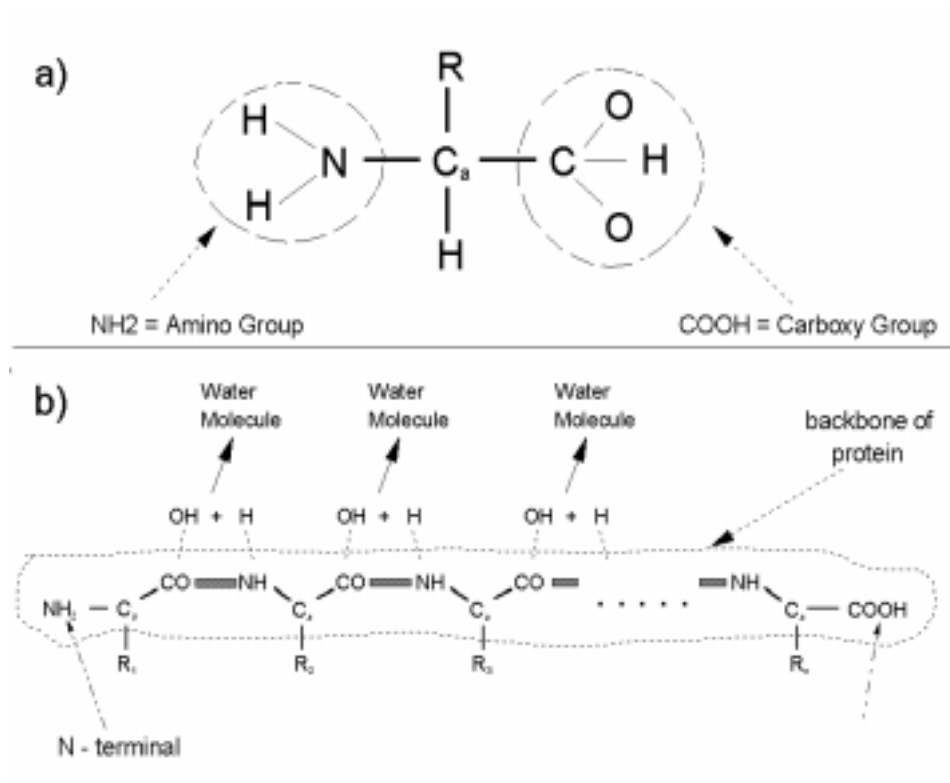


Figure 1.3: (a) All amino acids have the same underlying structure. There is a central carbon atom, called *alpha-Carbon* (symbolized here as C_α) and 4 chemical groups attached to C_α : a hydrogen atom, the nitrogen of an amino group, the carbon of a carboxyl group and an amino acid-specific side chain (symbolized here with the letter R). There are twenty different possible side chains, giving rise to one amino acid each. All the atoms in the amino acid are connected through covalent bonds.

(b) A protein formed by n amino acids bound together with *peptide bonds*. Peptide bonds are formed when the carboxyl group of one amino acid interacts with the amino group of another amino acid. A by-product of this interaction is the release of a water molecule (the carboxyl loses an oxygen and a hydrogen while the amino group contributes a hydrogen). Peptide bonds are shown in the figure with solid gray lines.

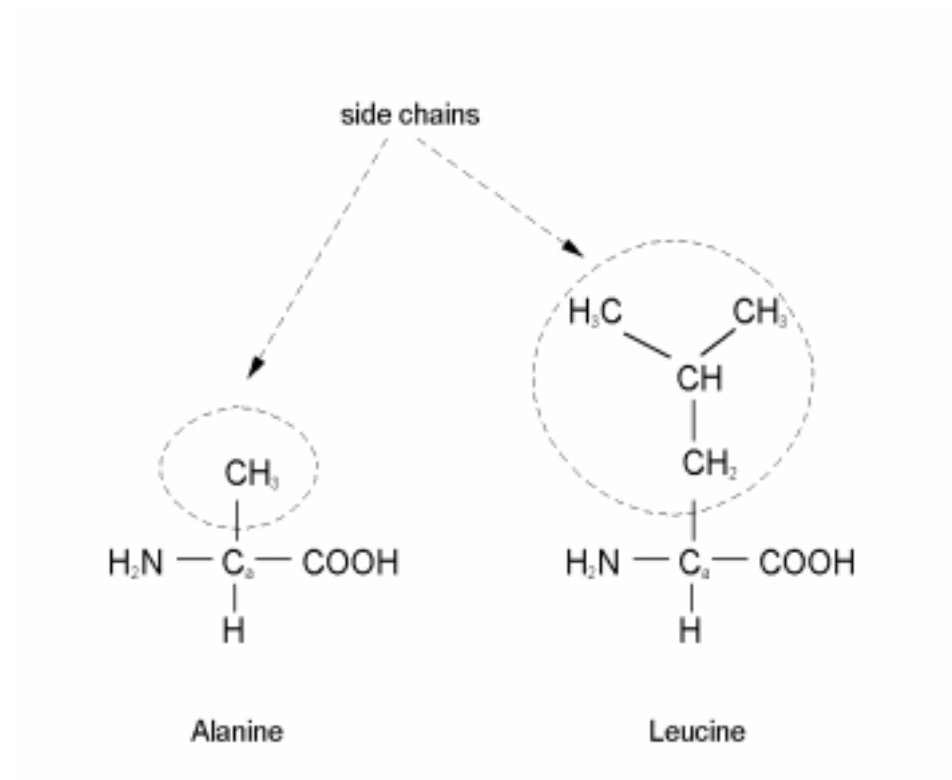


Figure 1.4: The chemical make up of the amino acids Alanine and Leucine.

Amino Acid Name	Three Letter Code	One letter Code	Chemical Class
Alanine	Ala	A	H
Valine	Val	V	H
Phenylalanine	Phe	F	H
Proline	Pro	P	H
Methionine	Met	M	H
Isoleucine	Ile	I	H
Leucine	Leu	L	H
Glycine	Gly	G	H
Aspartic Acid	Asp	D	C
Glutamic Acid	Glu	E	C
Lysine	Lys	K	C
Arginine	Arg	R	C
Serine	Ser	S	P
Threonine	Thr	T	P
Tyrosine	Tyr	Y	P
Histidine	His	H	P
Cysteine	Cys	C	P
Asparagine	Asn	N	P
Glutamine	Gln	Q	P
Tryptophan	Trp	W	P

Table 1.1: The 20 amino acids along with their three and one letter codes. The last column is used to classify the amino acids according to their chemical properties. The following character codes are used for that column: H = Hydrophobic, C = Charged, P = Polar.

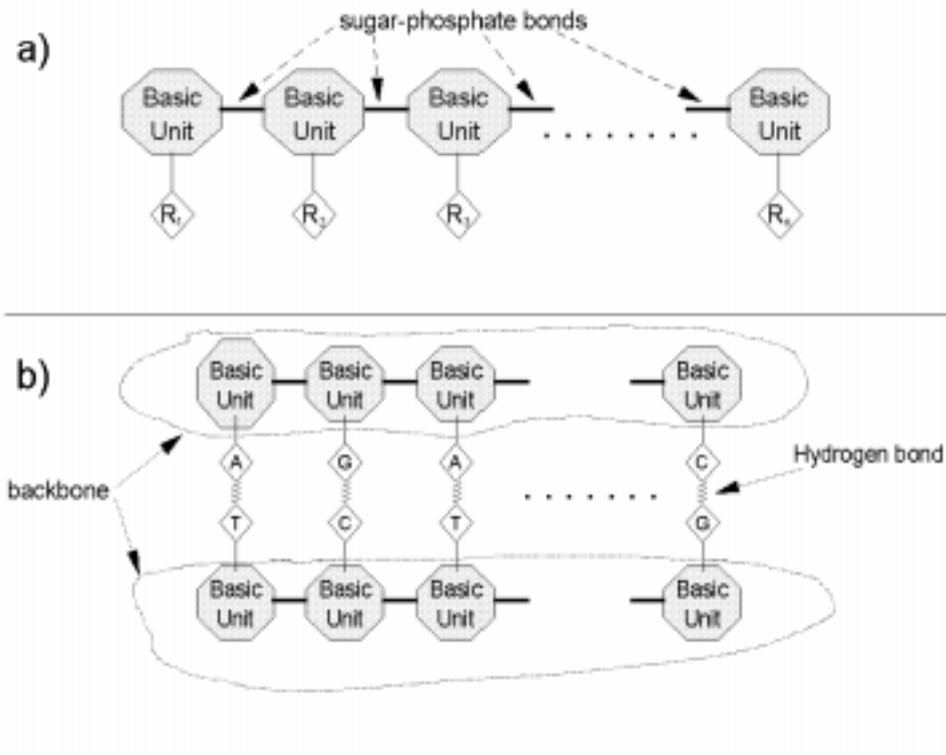


Figure 1.5: (a) Polymeric chains of nucleotides are formed when the basic units of nucleotides are bound together through sugar–phosphate bonds. The R_i indicate the side groups of the nucleotides.

(b) The actual DNA molecule is comprised of two parallel chains, connected through hydrogen bonds between side groups. Not all possible nucleotide pair combinations are possible: the rule is that A can only bind with T and C can only bind with G.

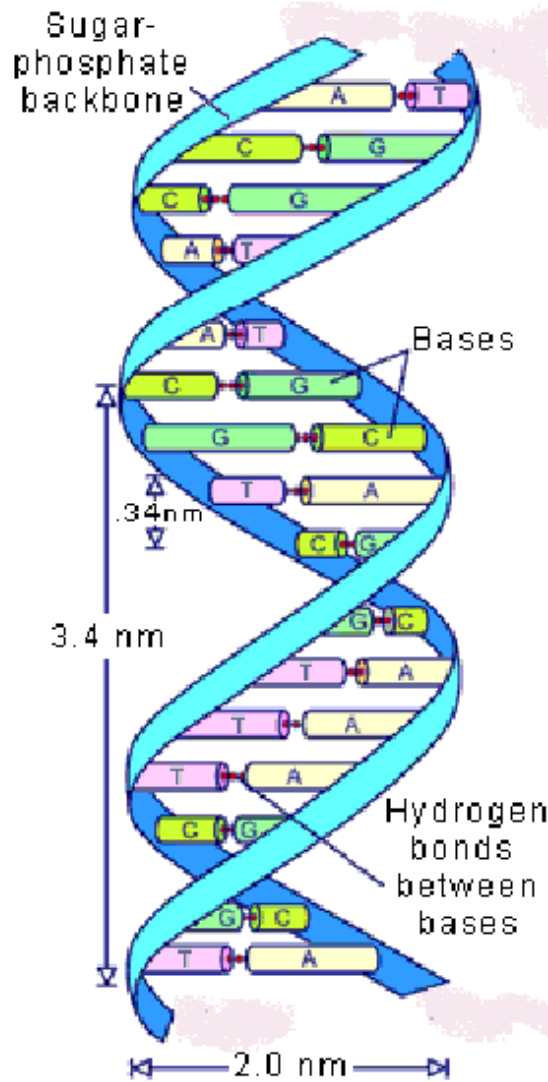


Figure 1.6: Three dimensional representation of the DNA double helix. The two parallel DNA strands form a right-handed helix, held together by the hydrogen bonds between complementary nucleotides.

		Second Position of Codon				
		I	C	A	G	
F i r s t P o s i t i o n	T	TTT Phe [F]	TCT Ser [S]	TAT Tyr [Y]	TGT Cys [C]	T
		TTC Phe [F]	TCC Ser [S]	TAC Tyr [Y]	TGC Cys [C]	C
		TTA Leu [L]	TCA Ser [S]	TAA Ter [end]	TGA Ter [end]	A
		TTG Leu [L]	TGG Ser [S]	TAG Ter [end]	TGG Trp [W]	G
	C	CTT Leu [L]	CCT Pro [P]	CAT His [H]	CGT Arg [R]	T
		CTC Leu [L]	CCC Pro [P]	CAC His [H]	CGC Arg [R]	C
		CTA Leu [L]	CCA Pro [P]	CAA Gln [Q]	CGA Arg [R]	A
		CTG Leu [L]	CCG Pro [P]	CAG Gln [Q]	CGG Arg [R]	G
	A	ATT Ile [I]	ACT Thr [T]	AAT Asn [N]	AGT Ser [S]	T
		ATC Ile [I]	AOC Thr [T]	AAC Asn [N]	AGC Ser [S]	C
		ATA Ile [I]	ACA Thr [T]	AAA Lys [K]	AGA Arg [R]	A
		ATG Met [M]	ACG Thr [T]	AAG Lys [K]	AGG Arg [R]	G
G	GTT Val [V]	GCT Ala [A]	GAT Asp [D]	GGT Gly [G]	T	
	GTC Val [V]	GCC Ala [A]	GAC Asp [D]	GGC Gly [G]	C	
	GTA Val [V]	GCA Ala [A]	GAA Gln [E]	GGA Gly [G]	A	
	GTG Val [V]	GCG Ala [A]	GAG Gln [E]	GGG Gly [G]	G	

Figure 1.7: The mapping of nucleotide triplets to amino acids used by living organisms.

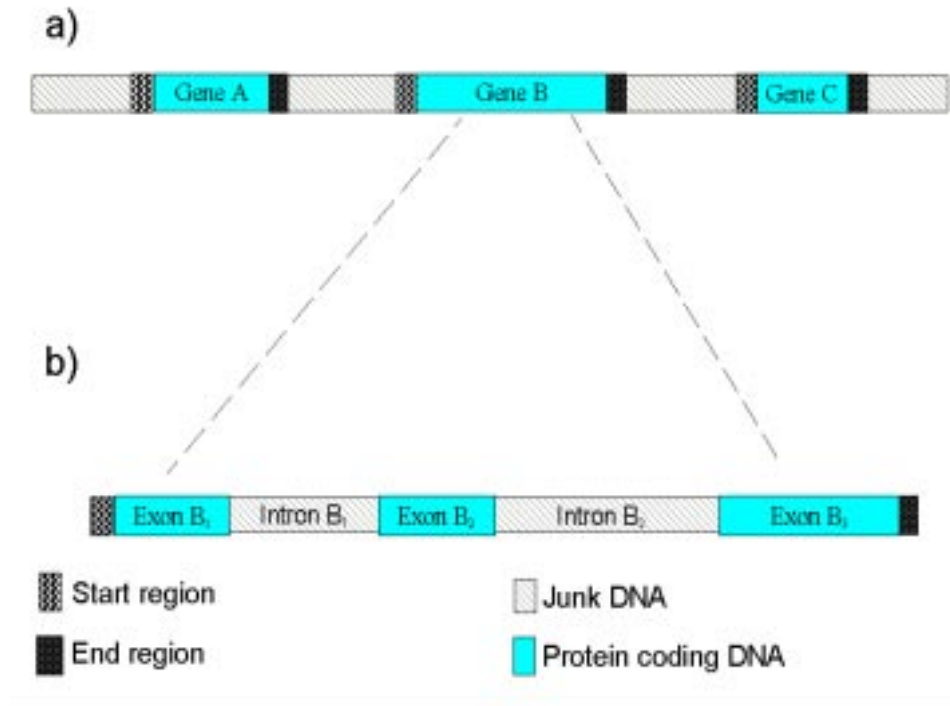


Figure 1.8: (a) Different regions of DNA have different functionalities. The so-called *coding regions* or *genes* are consecutive stretches of nucleotides that contain the codons for given proteins. The area covered by a gene is delineated by a *start* and an *end* region. Furthermore, there exist large chunks of DNA (called *junk DNA*) that seem to have no function at all.

(b) Genes are themselves some times composed of both coding regions (called *exons*) and non-coding regions (called *introns*). Here, the gene *B* of part (a) is magnified, revealing its composition of both exons and introns.

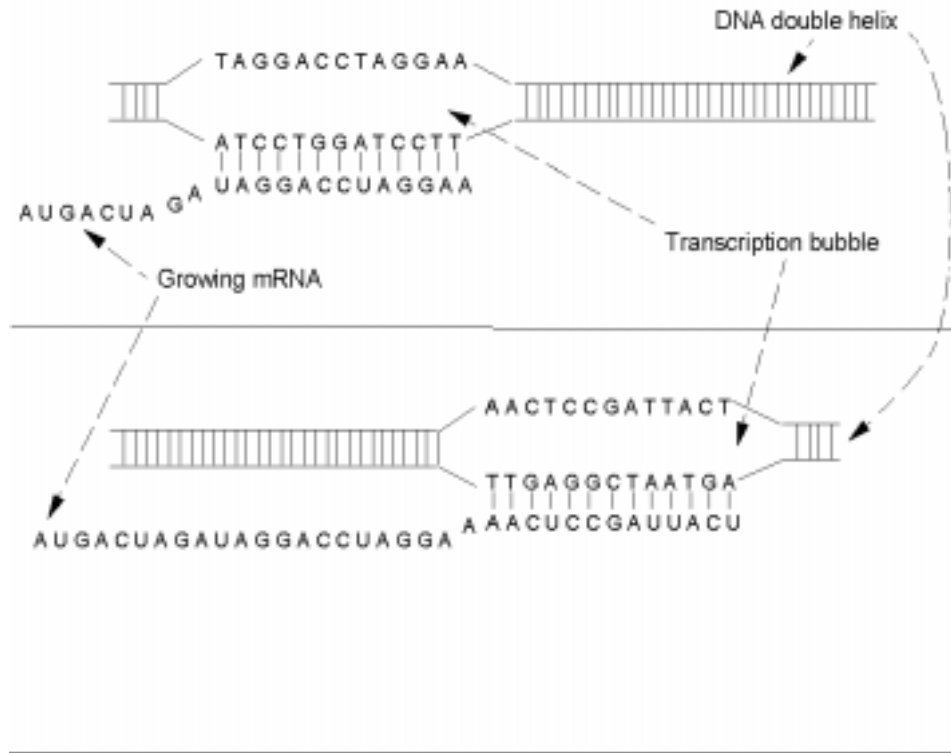


Figure 1.9: Two snapshots of the process of transcribing a gene to mRNA. The RNA polymerase (not shown here) marches over the gene. As it passes through, it locally unzips the two strands making the DNA helix, temporarily creating a *transcription bubble*. One of the two strands is then used as a template in order to elongate the mRNA molecule being built by the polymerase. When done, the RNA polymerase closes the current transcription bubble and proceeds to create a new one.

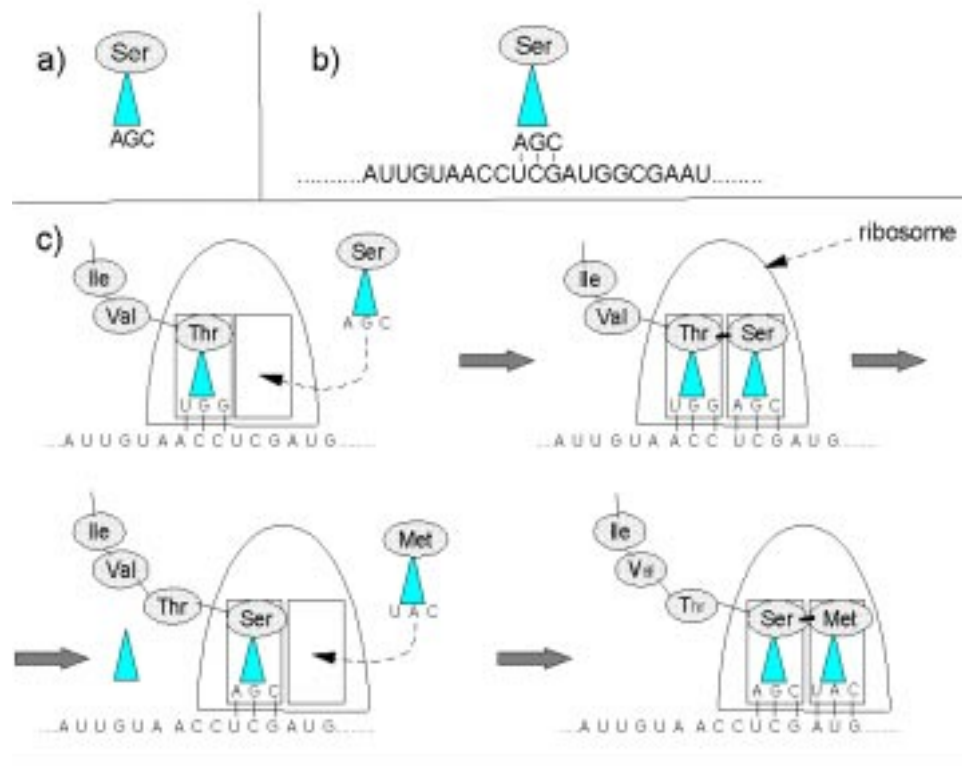


Figure 1.10: (a) Example of a tRNA molecule. At one end of the RNA stem there is the anticodon AGC which is complementary to the codon TCG that codes for the amino acid Serine. This amino acid is attached to the other end of the tRNA.

(b) The anticodon matches mRNA codons coding for Serine.

(c) Translation of mRNA into protein by a ribosome. The protein chain is gradually built as the ribosome decodes one by one the codons of the underlying mRNA molecule. When a tRNA enters the rightmost cavity of the ribosome, it is connected (thick black line) to the partial chain which is attached to the tRNA in the first cavity. Then the stem of the first tRNA gets disconnected from the chain, it gets discarded from the ribosome and the second tRNA moves to the leftmost cavity, dragging along the entire mRNA molecule. As a result, the rightmost cavity is left free so that the translation of the next codon can begin.

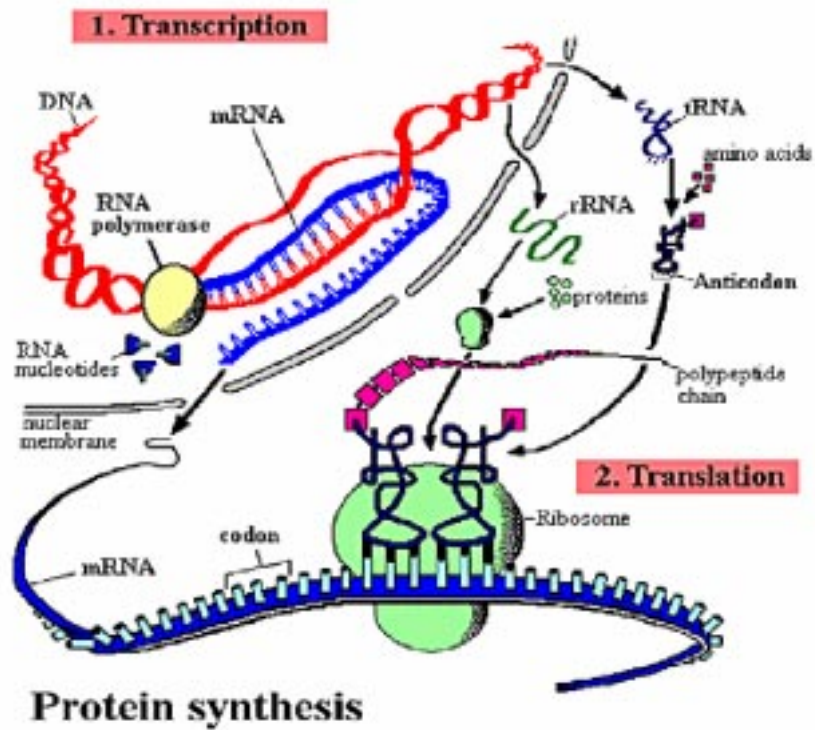


Figure 1.11: A schematic representation of the transcription and the translation processes.

DNA REPLICATING ITSELF

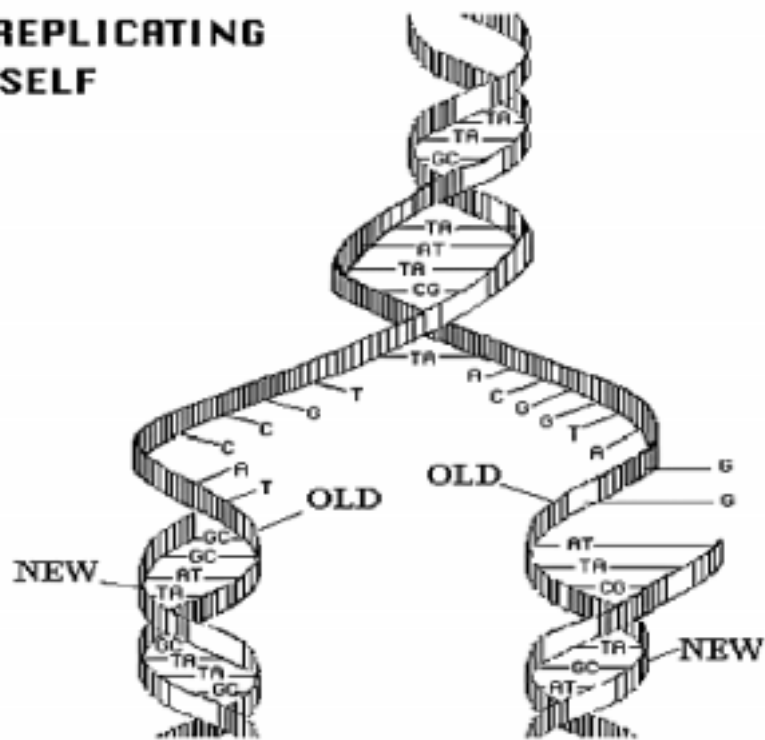


Figure 1.12: Replication of the DNA double helix.

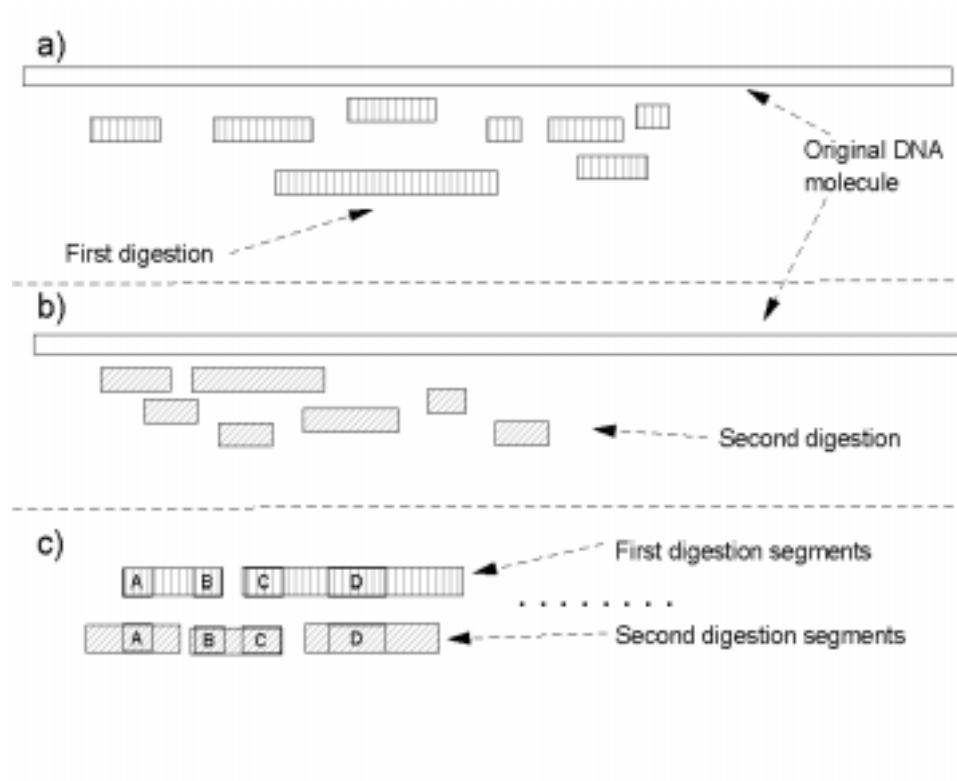


Figure 1.13: A schematic representation of the process of sequencing a long DNA molecule. First, the original DNA molecule is digested in two alternative ways ((a) and (b)). Then the pieces of each digestion are sequenced independently. Finally, in (c) a *tiling path* is built combining the areas shared by the various pieces (areas *A*, *B*, *C* and *D*).

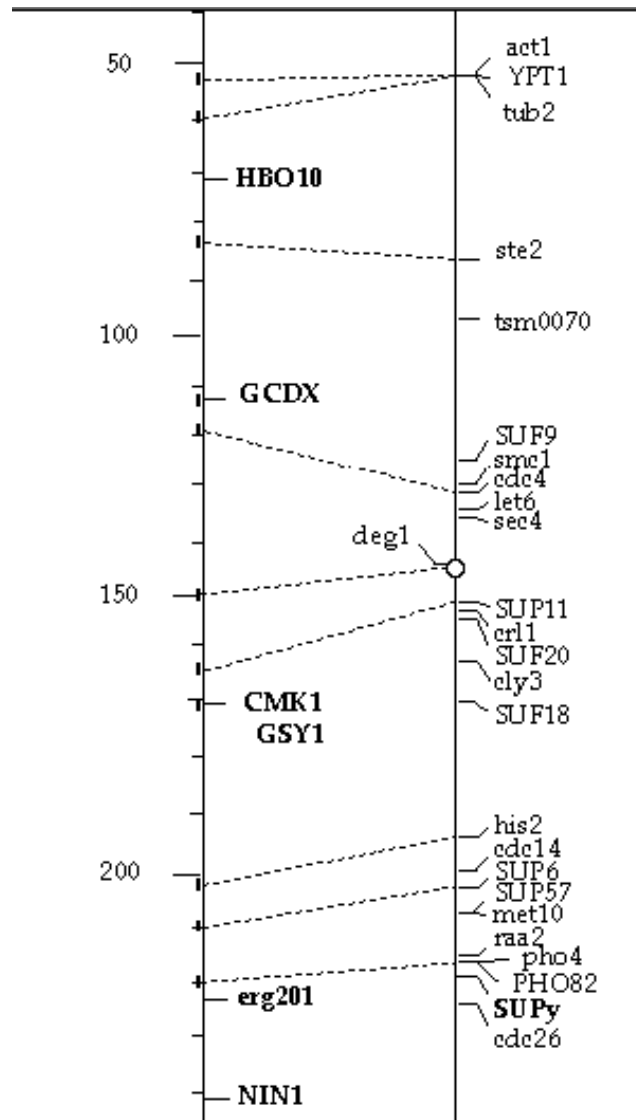


Figure 1.14: Genetic (right) map of chromosome VI of the organism *Saccharomyces cerevisiae*, a unicellular eucaryote. Ticks on the rightmost line indicate genes (the names of these genes are also given). The genes are shown in their relative order of appearance on the chromosome. For those genes for which their actual physical location is known, this location is shown on the leftmost line. In this line, numbers indicate distances (in Kbps) from the beginning of the gene. Dotted lines connect genes (on the right) with their physical location (on the left). Also shown on the leftmost line are genes of known location for which there is not enough information in order to place them on the genetic map.

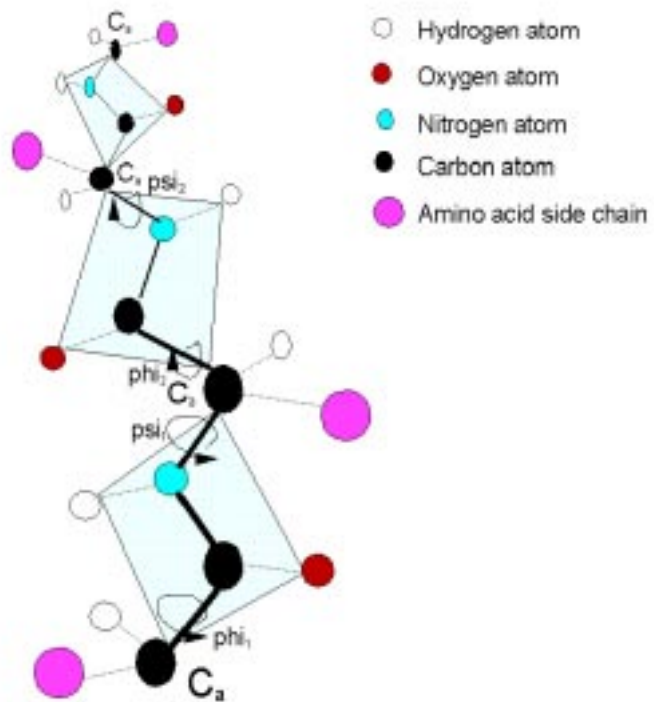


Figure 1.15: The three dimensional structure of protein backbone.

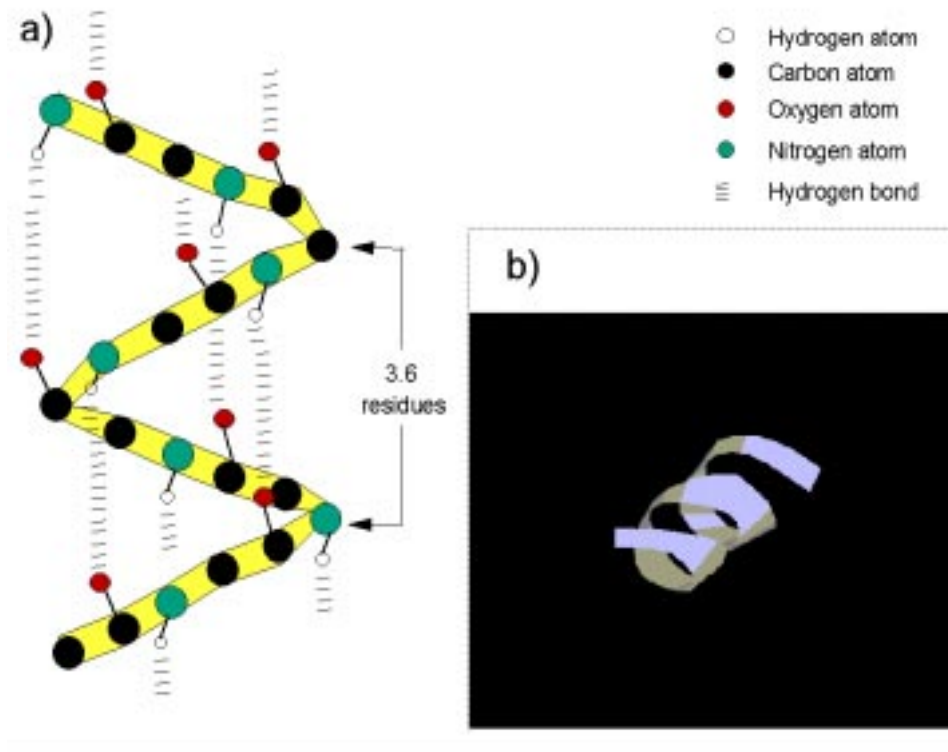


Figure 1.16: (a) An α -helix is formed by hydrogen bonds between hydrogen and oxygen atoms. It looks like a spring with one turn every 3.6 amino acids. The total number of turns varies from α -helix to α -helix.

(b) A three dimensional rendering of an α -helix. A ribbon is used for representing the backbone of the helix.

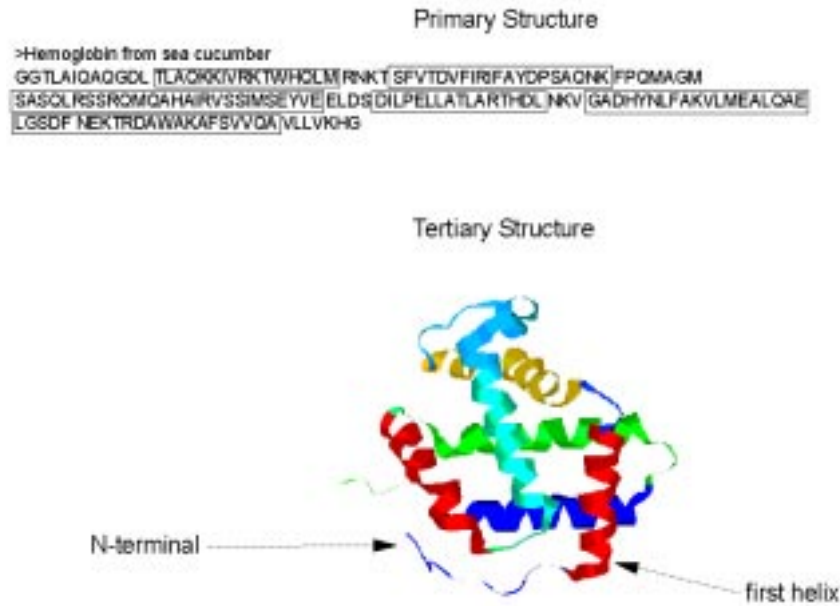


Figure 1.17: The primary and tertiary structure representation of a hemoglobin protein. The tertiary structure is composed of six α -helices. Each one is represented by a distinct color: red for the first helix (the one closest to the N-terminal of the protein), yellow for the second helix and so on. On the primary structure, the amino acid regions that correspond to the helices are shown surrounded by boxes.

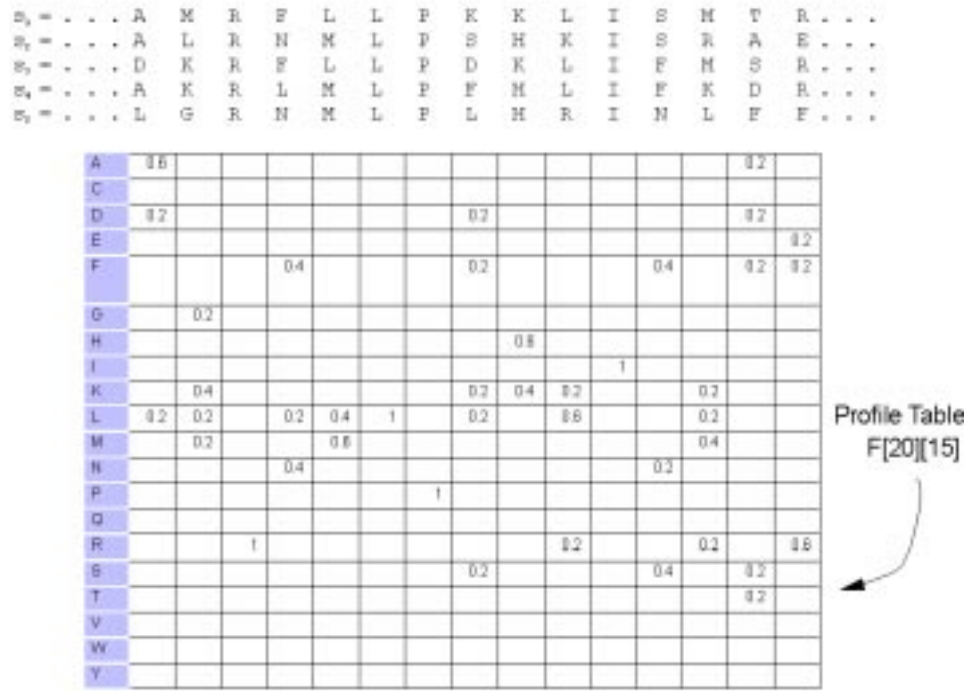


Figure 1.18: A profile can be built from any alignment of protein regions. In this example, five sequences (s_1, s_2, s_3, s_4, s_5) are aligned along a region of 15 amino acids. The local alignment gives rise to a table (the profile) with one row per amino acid and one column per alignment position. For any given column every amino acid is assigned a weight indicating its contribution to the alignment column under consideration. In this example we use as weight the normalized frequency of appearance of the amino acid in the alignment column (frequencies are normalized so that they sum up to 1). Profiles are used to characterize new proteins. Given a profile table $F[20][n]$ for an alignment of length n (in this example, $n = 15$) and an arbitrary sequence s , it is possible to assign the following weight to every sequence position i : $W(i) = \sum_{j=0}^{n-1} P[s[i+j]][j]$. Positions with a high scores can then be said to mark the beginning of an instance of the profile.

Chapter 2

Pattern Discovery Algorithms

This chapter provides an in-depth discussion of the pattern discovery problem. A precise definition of the problem is given and its computational complexity is analyzed. We show that the complexity is dependent on the definition of the concept of a pattern and a number of existing algorithms handling the discovery problem for several different pattern classes are presented. A new pattern discovery algorithm called **TEIRESIAS** is introduced and its design and performance are discussed in detail. We conclude with applications of **TEIRESIAS** in a number of test cases, exhibiting the utility of the algorithm in the biological domain.

2.1 Problem Description

As discussed in Section 1.2, patterns provide appropriate representations of conserved regions in biosequences. In most cases one is given a set of sequences (DNA or proteins) and is looking for patterns that appear in some minimum number (or percentage) of these sequences.

The exact definition of a pattern varies from algorithm to algorithm. In general, a pattern is a member of a well defined subset \mathcal{C} of all the possible regular expressions over Σ^1 (the set \mathcal{C} is called a *pattern language*). Being a regular expression, every

¹In the sequel, Σ is used to denote the alphabet from which the biosequences are drawn, i.e. Σ is either the set of nucleotides or the set of amino acids. Unless explicitly

pattern P defines a language $\mathcal{L}(P)$ in the natural way: a string belongs to $\mathcal{L}(P)$ if it is recognized by the automaton of P . A sequence $s \in \Sigma^*$ is said to “match” a given pattern P if s contains some substring that belongs to $\mathcal{L}(P)$.

For an illustration, consider the following set of strings over the english alphabet:

$$S = \{\text{LARGE, LINGER, AGE}\} \tag{2.1}$$

In this case the pattern “L..GE” has *support* 2 since it is matched by the first two strings of S (‘.’ is called the *don't-care* character and is used to indicate a position that can be occupied by an arbitrary alphabet character). The term *support* denotes the number of input strings matching a given pattern. As another example, the pattern “A*GE” has also support 2 (it is matched by the first and the last strings). Here, the character ‘*’ is used to match substrings of arbitrary length.

The pattern discovery problem is defined as follows:

The Generic Pattern Discovery Problem

Input: A set $S = \{s_1, s_2, \dots, s_n\}$ of sequences from Σ^* , and an integer $k \leq n$.

Output: All the patterns in \mathcal{C} with support at least k in the set S .

The key factor dominating the computational complexity of the pattern discovery problem is the “richness” of the pattern language \mathcal{C} . Ideally, one wants to define \mathcal{C} to be expressive enough for describing all the biologically important features of the sequences under examination. In other words, if a number of sequences contain regions which are biologically related we would like to have a pattern $P \in \mathcal{C}$ so that all these strings are members of $\mathcal{L}(P)$. Unfortunately, the amount of expressiveness allowed in \mathcal{C} directly impacts the computational demands of the problem. In the simplest case, when $\mathcal{C} = \Sigma^*$, the problem of finding all patterns in \mathcal{C} with a given minimum support can be solved in linear time using *generalized suffix trees* [56]. In almost every other case though, the class \mathcal{C} is expressive enough

stated otherwise, the discussion that follows applies to both DNA and protein sequences.

to render the problem NP-hard (the hardness result can be usually shown by a reduction from the *longest common subsequence* problem [34, 73]). Examples of pattern languages (in order of increasing expressiveness) are the following:

- $\mathcal{C} = \Sigma^*$: This is the simplest possible pattern language, comprised by simple strings over the residue alphabet.
- $\mathcal{C} = (\Sigma \cup \{'.\})^*$: here the character ‘.’ (the *don't-care* character) indicates a position that can be occupied by an arbitrary residue.
- $\mathcal{C} = (\Sigma \cup R)^*$: $R = \{R_1, \dots, R_n\}$ is a collection of sets $R_i \subseteq \Sigma$ and the use of a set R_i in a pattern indicates an amino acid position that can be occupied by *any* residue in R_i . For example, let $R_1 = \{A, F, L\}$ be a set of amino acids. Then the language $\mathcal{L}(GR_1K)$ contains the amino acid strings of length 3 that start with a glycine (G), end with a lysine (K) and have in their middle position any of the amino acids alanine (A), phenylalanine (F) or leucine (L). (The above mentioned pattern would actually be written as $G[AFL]K$.)
- $\mathcal{C} = (\Sigma \cup \{ '* \})^*$: the character ‘*’ is the Kleene star of Σ and is matched by strings of arbitrary length.
- $\mathcal{C} = (\Sigma \cup X)^*$: X is a set containing components of the form $x(i, j)$ with $i \leq j$ which indicate *flexible gaps* of constrained lengths; $x(i, j)$ is matched by any string of length between i and j inclusive. Notice that every pattern in this category can be thought of a union of patterns of the type $(\Sigma \cup \{'.\})^*$. For example,

$$Ax(1, 3)B = \{A.B, A..B, A...B\}.$$

In order to appreciate the nature of the difficulty that the richness of \mathcal{C} imposes on the pattern discovery problem, the following example is instructive. Consider the following class of patterns:

$$\mathcal{C} = \Sigma(\Sigma \cup \{'.\})^*\Sigma$$

i.e. all the patterns that start and end with an alphabet character and contain an arbitrary number of alphabet characters and don't-cares in between. Let $s = a_1a_2 \dots a_n$, with $a_i \in \Sigma$. For simplicity, we assume that $i \neq j \Rightarrow a_i \neq a_j$. Define now the set of sequences

$$S = \{s_1, s_2, \dots, s_n\}, \quad s_i \in \Sigma^* \quad (2.2)$$

where every s_i is obtained from s by replacing the character a_i in s with a character $a'_i \in \Sigma$. Furthermore, assume that:

- $\forall i, j : a'_i \neq a_j$.
- $\forall i \neq j : a'_i \neq a'_j$.

Then for every $Q \subseteq S$ with $|Q| = m, 2 \leq m \leq (n - 1)$, there exists a *unique* pattern $P_Q \in \mathcal{C}$ such that P_Q is matched by *all* the sequences in Q and by *no* other sequence in S . Figure 2.1 gives an example of how P_Q can be obtained by aligning all the sequences of Q one under the other and substituting every column containing more than one character with the don't-care.

In conclusion:

Given an integer $k, (2 \leq k \leq (n - 1))$, the number of \mathcal{C} patterns that have support k or more in S are at least

$$\sum_{m=k}^{n-1} \binom{n}{m}$$

As the above example shows, there can be pathological cases where the number of existing patterns is exponential in the size of the input (in Section 2.3.6 we will give a more detailed characterization of such cases). Just reporting these patterns (let alone generating them) is a formidable task.

In order to put the above observation in perspective, we will borrow some terminology from the area of *machine learning*. The selection of a particular pattern language \mathcal{C} by a pattern discovery algorithm introduces a *search space* comprised by all the patterns that belong in \mathcal{C} . Any particular instance of the problem (that

is any set S of input sequences along with a minimum support requirement k) defines a corresponding *solution space*, namely the subset of all the patterns of \mathcal{C} which have the requested minimum support in the given input set of sequences. Unlike the search space (which is usually huge) the solution space for a given instance is, in many cases, of moderate size (except when dealing with pathological inputs, like the one given on the example above). The challenge facing any pattern discovery algorithm is the efficient computation of the solution space while avoiding the *redundant exploration* [88] of the search space.

As we will see in the next section, pattern discovery algorithms can be categorized according to the strategy they use to explore the search space introduced by the pattern language that they adopt. Some of them avoid a complete exploration altogether by exploiting alternative, approximation techniques which are guaranteed to work “fast” but do not necessarily find *all* the patterns for *every* given input. Other approaches (the so called “exact” or “combinatorial” algorithms) just accept the hardness of the problem and proceed head-on to find all possible patterns. Such algorithms are bound to be inefficient (space and/or time-wise) on some inputs; however, depending on the domain at hand, their overall performance (amortized over all “reasonable” inputs) might be quite satisfactory. In such cases it becomes very important to use a number of heuristics in order to exploit any structure in the search space. Furthermore, extra speed can be usually gained by *parameterizing* the pattern language. This is usually left to the discretion of the user by providing him/her with the ability to appropriately set a number of parameters that decide the structure of the patterns to be looked for. A typical example is to allow the user to define the maximum length that a pattern can have. Providing the user with that kind of control over the search space is not unreasonable since an expert user can apply domain knowledge to help a program avoid looking for patterns that are meaningless or impossible for the domain at hand. In fact, this expert knowledge is usually an integral part (in the form of various heuristics) of most of the pattern discovery algorithms that exist in the literature. The disadvantage of this approach is that most of these algorithms are usually not applicable outside the biology domain.

In the following section we survey a number of algorithms used for the discovery of patterns on biomolecular data. Both approximation and exact methods are discussed. In evaluating the algorithms presented it is important to keep in mind that the traditional computer science quality assessment criteria (i.e. the time and space complexity) are just one component of the overall value of a methodology (this is particularly true for the exact algorithms, since the problem is NP-hard). There are also other criteria which might be more relevant. Some of them are:

- The pattern language \mathcal{C} used. In general, it is desirable to have as expressive a class \mathcal{C} as possible. The price for the increased expressiveness is usually paid in terms of time/space efficiency.
- The ability of the algorithm to generate all qualified patterns. As mentioned earlier, some approximation algorithms can achieve increased performance by sacrificing the completeness of the reported results. Depending on the domain of the application and the quality of the patterns discovered, this might or might not be an acceptable trade-off.
- The *maximality* of the discovered patterns. Consider for example the instance of the input set S in (2.1). In this case “L...E” is a perfectly legal pattern. It is not however maximal, in the sense that the pattern “L..GE”, while more specific, still has the same support as “L...E”. Reporting patterns which are not maximal clutters the output (making it difficult to identify the patterns that are really important) and can severely affect the performance of an algorithm. It is thus extremely important for a pattern discovery algorithm to be able to detect and discard non-maximal patterns as soon as possible.

These criteria are used in the next section for the evaluation of the algorithms presented therein.

2.2 Previous Work

The pattern discovery algorithms can, in general, be categorized in either of the following two classes [18]: *string alignment* algorithms and *pattern enumeration*

algorithms. Below we present a short survey of both categories. The list of algorithms discussed is certainly not exhaustive but it highlights the main trends in the respective classes.

2.2.1 Alignment Algorithms

Algorithms in this class use *multiple alignment* of the input sequences as the basic tool for discovering the patterns in question. There exist more than one ways to define the mutual global alignment of several input sequences. Below, we provide a definition based on the concept of the *consensus sequence* as (i) it captures the “essence” of the problem, (ii) it is simple and clean and (iii) it is sufficient for the purpose of the discussion in this section. A more formal definition along with an extensive exposition can be found in [41]. We will revisit this problem in some more detail in Section 5.2.

The Multiple Sequence Alignment (MSA) Problem

Given a set $S = \{s_1, s_2, \dots, s_n\}$ of strings over an alphabet Σ and a number of *edit operations* (mutation of a character into another character, deletion of a character, insertion of a character etc.), a multiple alignment of the strings in S is defined as:

- a string $w \in \Sigma^* \cup \{-\}$, called a *consensus sequence*. The character ‘-’ is assumed not to be part of Σ and is used to denote a “gap” introduced by an insertion.
- for each $1 \leq i \leq n$, a sequence of edit operations for transforming s_i into w .

Multiple string alignments are usually depicted graphically, as in Figure 2.2. In this representation the edit operations that have been applied on the input sequences are described implicitly by the positioning of the input strings relative to the consensus sequence. It is worth pointing out that alignments do not *explicitly* generate all the existing patterns (except for patterns which are shared by *all* the input sequences). They do, however, highlight the conserved areas. Closer examination of such areas can then produce the desired patterns ([79] describes

an algorithm for producing all the patterns with a given minimum support from blocks of aligned sequence regions). Figure 2.3 shows an alignment of actual proteins.

As given above, the definition of the MSA problem is incomplete. In particular, what is missing are specific restrictions which guarantee that the produced alignment is biologically relevant, i.e. that it correctly highlights conserved regions on the aligned sequences. In the absence of such restrictions any consensus string and any set of edit operations could in principle be used, thus allowing for biologically non-sensical alignments. Biological relevance of the alignments is imposed by assigning costs to the various edit operations (through the use of biologically sensible cost matrices like the PAM [25] and BLOSUM [45] families), and by defining an objective function over the space of all possible multiple alignments. The problem then becomes one of *optimally* aligning the input sequences, i.e. selecting a consensus sequence which will minimize the objective function.

A complete description of the problem is outside the scope of this dissertation. What is important to note, though, is the fact that under any reasonable definition of the objective function the problem of finding an optimal sequence alignment is NP-hard [109]. As a result, most algorithms in this class resort to heuristics that produce suboptimal alignments, thus trading-off enhanced execution speed with results that may not be complete. There are also other problems related to global string alignment (e.g. the problem of domain swapping — see Figure 2.4) which further complicate the generation of a complete list of patterns for a given input set. In general, using multiple string alignment for the discovery of patterns can be effective only when the strings aligned share global similarities [95].

In the domain of Biology, most of the algorithms dealing with the MSA problem begin with the pairwise alignment of all possible pairs of input sequences and then apply a number of heuristics in order to approximate an optimal global alignment. Martinez in [51] starts by performing and scoring all such pairwise alignments. Then an ordering of the input strings is generated, based on the alignment scores (the intention is for strings that are similar to be placed close together in that ordering). The final multiple alignment is built in a piece-wise manner, by

traversing the ordered list of the input strings: each time a new string is added, the old alignment is modified (by adding insertions where appropriate) so that the character-to-character matches of the original, pairwise alignments are preserved.

A slightly different approach is pursued by Smith and Smith [96]. Again, the starting point is generating and scoring all possible pairwise alignments. Scoring is performed based on a partition of the amino acid alphabet into *amino acid class covering (AACC)* groups (similar to those defined by the last column of Table 1.1), based on the physicochemical properties of the amino acids. Using the scores obtained during the first step, a binary *dendrogram* is built which, in effect, clusters together those of the input sequences which are similar. Then the internal nodes of the dendrogram are traversed bottom-up. At the first level of traversal, both children of a node are leaves (the original input sequences). Such nodes are replaced by patterns, based on the alignment of their children; aligned columns that contains more than one residues are represented by the smallest AACC group that contains all of these residues (if none exists, a don't care character is used). The same procedure is propagated towards the root of the dendrogram pattern that fare further up appearing as children of the internal nodes. Upon termination, each internal node has been labeled by a pattern which is shared by all the input sequences that are leaves of the subtree rooted at that internal node.

Another algorithm employing multiple string alignment as the main tool for the discovery of patterns is *Emotif*[79]. Here, along with the set S of the input strings, the user also provides a collection $R \subseteq 2^\Sigma$ of subsets of the alphabet Σ . A pattern can have at a given position any element $E \in R$ and a character c of an input string matches that pattern position if $c \in E$. The set R provides a generalization of the AACC groups used in [96]. *Emotif* assumes that a multiple alignment of the input strings is available. The alignment is used to guide the generation of the patterns in a recursive way: at each point a subset S' of the original set S is being considered (originally $S' = S$) and a particular column in the alignment of the strings in S' (originally the first column). Also, there is a pattern P currently under expansion; originally, P is the empty string. The pattern P is expanded to $P' = PE$ where $E \in R$ contains at least one of the characters found in the strings

of S' at the alignment column under consideration. The expansion proceeds as long as the new pattern has sufficiently large support and is not redundant, i.e. does not appear in the same sequences where a previously generated pattern has appeared. At the next expansion step, the set S' will comprise those strings that match the new pattern P' and the next column of the alignment will be considered.

A different heuristic is proposed by Roytberg [87]. Although his method does not directly use alignment, it works in a way reminiscent of the other algorithms in this class in that it gets information about potential patterns by pairwise comparisons of the input strings. One of the input sequences is selected as the *basic* sequence and is compared with all other sequences for similar segments (the notion of similarity is a parameter of the algorithm). A segment of the basic sequence gives rise to pattern if at least k sequences (the minimum required support) have segments similar to it. The major drawback of this method is that it is crucially dependent on the selection of the basic sequence. This drawback can be partially offset by employing repeated runs of the algorithm, each one with a different basic sequence.

2.2.2 Pattern Enumeration Algorithms

This class contains algorithms that enumerate all (or some part of) the solution space and then verify which of the patterns generated during this enumeration process have the required support. Since such algorithms explore the space of patterns they tend to be exponential on the size of the largest generated pattern. In order to be efficient, they usually impose some restrictions on the structure of the patterns to be discovered.

The underlying idea used by all these algorithms is the following: start with the empty pattern and proceed recursively to generate longer and longer patterns. At each step enumerate all (or some) allowable patterns (i.e. those patterns that belong to the subclass \mathcal{C} of regular expressions treated by the algorithm) that have the current pattern as prefix. For every new pattern check its support. If it is above a threshold continue the expansion. If not just report the current pattern and backtrack to the previous step of the expansion procedure.

What differentiates the various algorithms is the class \mathcal{C} of patterns that they recognize, the efficiency with which they implement the pattern expansion and their ability to quickly detect and discard patterns which are not maximal.

One of the first algorithms in this class [98] actually appeared as part of a program for multiple string alignment². The algorithm works by first locating substrings X common to *all* the input sequences, each substring having length at least L . For every such X the set of all possible *regions* is defined: each region is an n -tuple (p_1, \dots, p_n) (n is the number of input sequences) where p_i is an offset at which the i -th sequence matches X . These regions become vertices in a directed graph. If $R_X = (x_1, \dots, x_n)$, $R_Y = (y_1, \dots, y_n)$ are regions corresponding to the substrings X and Y then a line from R_X to R_Y is added in the graph if the two regions are non-overlapping and R_X comes before R_Y . In other words, if for every i

$$x_i + |X| - 1 < y_i.$$

Finally the graph is traversed for a longest possible path (they give a particular function for assigning costs to the edges but any meaningful cost function could be used in its place).

There is a number of problems with this method. First, it only finds patterns supported by *all* the input sequences. Furthermore, in order for it to be efficient the parameter L for the minimum length of substrings to look for must be quite large. Otherwise a huge number of regions will be generated. As a result, patterns containing shorter substrings will remain unnoticed.

A more straightforward example of pattern enumeration appears in the work of Smith et. al. [95]. They proceed by enumerating all patterns containing 3 characters and having the form

$$c_1 x(i) c_2 x(j) c_3, \quad 0 \leq i \leq d_1, 0 \leq j \leq d_2,$$

where d_1 and d_2 are user-set parameters, c_i are amino acids and $x(i)$ indicates a *rigid gap*, i.e. a region that is matched by any string of length exactly i (the term

² Some programs approximate optimal string alignments by first locating patterns common to all strings and then using these patterns as the anchor points for the alignment.

rigid is used for wild card regions matched only by strings of a fixed length, in contrast with *flexible* gaps of the form $x(i, j)$, $i \leq j$, which are matched by any string with length between i and j). Each pattern thus generated is matched against all the input strings. If it has sufficient support all sequences where it appears are aligned along this pattern and the pattern is further expanded according to the alignment. The main disadvantage of this method is that it actually enumerates all possible patterns, making it hard to handle patterns with more than 3 characters and with big values for the parameters d_1 and d_2 .

Neuwald and Green [77] provide an improved extension of the previous method. Their algorithm allows the discovery of rigid patterns of arbitrary length (obeying some structural restrictions set up by user-defined parameters). Furthermore, they allow double character pattern positions of the form $[c_1c_2]$ which can be matched by either of the characters c_1, c_2 ; the number of these positions, though, must be kept to a minimum or the performance of the algorithm suffers. Their algorithm starts by enumerating all possible patterns with a given maximum length, given number of non-don't care positions and a maximum allowed number of double character positions. The enumeration is carried out efficiently in a depth-first manner using *blocks*, a special data structure that records, for every character, all offsets of the input strings that the character has a fixed distance from. The enumeration is further sped up by pruning the patterns that do not achieve sufficient support. Using statistical analysis, they keep just a portion of the patterns thus generated (those that are deemed "important"). At the final step, these patterns are combined into longer patterns by pairwise comparison: two patterns are expanded into a new one if they have adequate "compatible" appearances, i.e. if there exist enough sequences where the two patterns appear separated by a fixed displacement. This expansion operation is made possible because of the block structure representation of the patterns which contains the list of offsets at which the patterns appear.

Based on the work of Neuwald and Green above, Collins et. al. [57] present an even more powerful pattern discovery algorithm which allows for flexible patterns of the following form:

$$P = A_1x(i_1, j_1)A_2x(i_2, j_2) \dots A_{(p-1)}x(i_{(p-1)}, j_{(p-1)})A_p$$

where $i_k \leq j_k$. The A_i 's are character sets (consisting of one or more characters) and are called *components*. A component can be of one of two types: *identity* or *ambiguous*, depending on whether it contains one or more than one characters. The wild-card regions $x(i, j)$ are either rigid (if $i = j$ — such a region is written in the simpler form $x(i)$) or flexible if ($j > i$). In addition to the actual sequences, the input to the algorithm contains a host of other parameters that restrict the kind of patterns to look for (e.g. maximum pattern length, maximum number of components, maximum number of ambiguous components, the kind of ambiguous components allowed etc.) as well as the minimum support k for a pattern. The algorithm proceeds in two phases. In the first phase, the block structure of [77] is used to enumerate all patterns with length up to the maximum possible. As a preprocessing step, blocks $b_{i,R}$ are created for every allowable component R (be it either a single character or a set of characters). Every such block contains offsets in the input sequences. A particular offset is in $b_{i,R}$ if some character in R is at distance i from that offset. If only rigid patterns have been requested and P is the current rigid pattern at any given time, then they check all possible patterns P' of the form $P' = Px(j)R$ where $x(j)$ is a rigid wild-card region of length j and R is either a single character or an allowable ambiguous component. If B_P is the *offset list* for P (every pattern carries along the list of offsets where it appears within the input sequences), then

$$B_{P'} = B_P \cap b_{|P|+j+1,R}$$

where $|P|$ is the length of the pattern P (i.e. the length of every string matching P).

If, on the other hand, flexible patterns have also been allowed then a flexible pattern P is represented by the set $F(P)$ of all the rigid patterns that comprise it. For example, if

$$P = Dx(1,2)Ex(2,3)F,$$

then the set $F(P)$ consists of the fixed patterns

$$Dx(1)Ex(2)F,$$

$$Dx(1)Ex(3)F,$$

$$Dx(2)Ex(2)F,$$

$$Dx(2)Ex(3)F.$$

Every pattern $Q \in F(P)$ carries its own offset list.

In this case the current pattern P is extended into $P' = Px(i, j)R$ using all possible values of $i \leq j$. The block structure for P' is actually constructed by extending every pattern $Q \in F(P)$ into a pattern Q_l , ($i \leq l \leq j$) using the relation

$$B_{Q_l} = B_Q \cap b_{|Q|+l+1, R}$$

and the set $F(P')$ becomes

$$F(P') = \cup_{Q \in F(P), i \leq l \leq j} Q_l$$

In both cases (flexible regions allowed or not), further expansion of P' is avoided if the size of the offset list of P' is less than the minimum support k . If P' is a flexible pattern then the size of its offset list is simply the sum of the sizes of the offset list of all fixed patterns in $F(P')$.

In the second phase, the discovered patterns are further processed in a number of ways. Possibilities include replacing some don't care characters by an extended alphabet of ambiguous components, extending a pattern, etc.

A similar algorithm is described by Sagot and Viari in [89]. Their algorithm also allows ambiguous components but it only treats rigid gaps. Again, the user must define (among other parameters like the maximum pattern length) which ambiguous components to use as well as, for every ambiguous component, the maximum number of times it is allowed to appear in a given pattern. Their algorithm also proceeds by recursively enumerating all allowable patterns, in a way very similar to that of [57]. The entire process is made faster by the introduction of two heuristics.

First, an effort is made to avoid creating redundant patterns. Let P be the currently expanded pattern. If S and S' are both allowable ambiguous components with $S \subset S'$ and PS, PS' are both possible extensions of P and if the offset lists of PS and PS' are the same, then only the pattern PS is maintained – the expansion is pruned at PS' . This heuristic does not detect all non-maximal patterns:

because of the way that the algorithm builds the patterns, some redundancies will go unnoticed.

Second, a clever trick is used in order to reduce the input size. Instead of considering the ambiguous components specified by the user, they originally replace all of them by the don't care character. This greatly simplifies the pattern space to search and makes the entire discovery process much faster. After this step only the part of the input matching the (reduced) patterns discovered is maintained; it is on this subset of the input that the algorithm is rerun, now using the full pattern specification given by the user.

A different approach in enumerating and verifying potential patterns is proposed by Wang et. al. in [108]. The pattern language they treat is composed of m -component patterns of the form

$$P = X_1 * X_2 * \dots * X_m$$

where the number m is a user specified parameter. The *components* X_i are amino acid strings and the “*” stands for flexible gaps of arbitrary length.

The algorithm works in two phases. First, the components X_i of a potential pattern are computed. This is done by building a generalized suffix tree (GST) for *all* the suffixes of the input sequences. Each leaf corresponds to such a suffix and is labeled with the sequence containing that suffix. Every internal node u contains the number $count(u)$ of *distinct* sequences labeling the leafs of the subtree rooted at that node. So, if $locus(u)$ is the string labeling the path from the root of the GST to an internal node u , then $count(u)$ is the number of distinct sequences containing the substring $locus(u)$. The first phase ends by creating the set Q that contains all strings $locus(u)$ such that $count(u) \geq k$, where k is the minimum support.

The second phase verifies which m -component patterns (among those that can be built from the elements of Q) have the minimum support. This is a computationally demanding process since every possible pattern has to be compared against all the input strings. To speed things up, the space of all potential m -component patterns is pruned using the following statistical heuristic: first, a (small) random subset S' of the original set S of input sequences is chosen. For every potential

pattern P , its support k'_P within S' is computed. Then, using k'_P and arguments from sampling theory, the probability that the actual support of P in S is k or more is derived. If this probability is very small, then the pattern P is thrown away. At the end of the heuristic, patterns which are unlikely to have the required support in S will have been discarded. The remaining patterns are then verified by computing their support over the original input set S . Those that appear in at least k of the input sequences are reported in the final results.

The use of the above statistical heuristic makes it possible to treat sets S with many sequences (although it entails the possibility of not detecting some important patterns that are not lucky enough to pass the statistical test). Furthermore, the approach is really effective only if the minimum support k is comparable to the number of sequences in S . Another drawback of the algorithm is that its performance deteriorates very fast as the parameter m increases. Finally, there is no provision for the detection of redundant patterns.

2.2.3 Conclusions

Each of the two classes of algorithms described above have their pros and cons. By permitting insertions and deletions, string alignment methods can locate flexible patterns, a category of increased expressiveness (and, thus, complexity). Also, by using fast approximations to optimal multiple string alignments, patterns can be quickly discovered. On the other hand, no matter how one chooses to assign cost for the allowable edit operations, there will always be inputs containing patterns that cannot be encapsulated by the optimal consensus sequence. This remains true even if near-optimal consensus sequences are considered. The problem becomes more acute for inputs where the sequences do not have global similarities. As a result, string alignment methods can be a viable approach only if

- the completeness of the reported results is not an absolute necessity,
- the input sequences can be clustered into groups that have global similarities, so that the alignment can produce ‘relevant’ consensus sequences.

The pattern enumeration algorithms, on the other hand, have the potential of generating all non-redundant patterns. The price to be paid for completeness, though, can be steep since a huge space of potential patterns has to be searched. This search problem is magnified many-fold if one allows for flexible patterns and/or patterns that permit multiple character positions. Furthermore, making sure that only maximal patterns are reported is a non-trivial task.

2.3 TEIRESIAS

The pattern discovery method that we propose in this dissertation belongs to the pattern enumeration class of algorithms. The main characteristics of our approach are:

1. The method is combinatorial in nature and reports *all* maximal patterns with a minimum support *without* enumerating the entire pattern space. This makes our approach efficient.
2. The patterns are generated in order of *maximality*, i.e. the maximal patterns are generated first. As a result, redundant patterns can be easily detected through comparisons to the patterns already generated. No costly post-processing or complicated bookkeeping is required.
3. Experimental results suggest that the algorithm is *output sensitive*, i.e. its running time is quasi-linear to the size of the produced output.
4. Unlike existing methods it can handle efficiently patterns of *arbitrary* length.

The algorithm operates in two phases: *scanning* and *convolution*. During the scanning phase, *elementary patterns* with sufficient support are identified. These elementary patterns constitute the building blocks for the convolution phase. They are combined into progressively larger and larger patterns until all the existing, maximal patterns have been generated. Furthermore, the order in which the convolutions are performed makes it easy to identify and discard non-maximal patterns.

2.3.1 Terminology and Problem Definition

Let Σ be the alphabet of residues at hand (e.g. the set of all amino acids or the set of all nucleotides). The pattern class handled by TEIRESIAS is

$$\mathcal{C}_{TEIR} = \Sigma(\Sigma \cup \{'.\}')^*\Sigma.$$

The language $\mathcal{L}(P)$ defined by a pattern $P \in \mathcal{C}_{TEIR}$ is the set of all strings that can be obtained from P by substituting each don't care by an arbitrary residue from Σ . For the pattern “A.CH..E” for example, the following oligopeptides are elements of $\mathcal{L}(A.CH..E)$:

ADCHFFE, ALCHSE, AGCHADE

For any pattern P , any substring of P that is itself a pattern is called a *subpattern* of P . For example, “H..E” is a subpattern of the pattern “A.CH..E”. A pattern P is called a $\langle L, W \rangle$ pattern (with $L \leq W$) if *every* subpattern of P with length W or more contains at least L residues.

Given a pattern P and a set of sequences $S = \{s_1, s_2, \dots, s_n\}$ we define the *offset list of P with respect to S* (or simply the offset list of P , when S is unambiguously implied) to be the following set

$$L_S(P) = \{(i, j) \mid \text{sequence } s_i \text{ matches } P \text{ at offset } j\}.$$

A pattern P' is said to be *more specific* than a pattern P if P' can be obtained from P by changing one or more don't care characters to residues or by appending an arbitrary string of residues and don't cares to the left or/and right of P . The following patterns are all more specific than the pattern “A.CH..E”

“AFCH..E”, “A.CHLE.K”, “SA.CH..E”

Notice that if a pattern P' is more specific than a pattern P then for every set S of input sequences $|L_S(P')| \leq |L_S(P)|$.

Given a set of sequences S , a pattern P is called *maximal* with respect to S if there exists no pattern P' which is more specific than P and such that $|L_S(P')| =$

$|L_S(P')|$. Conversely, if P is not maximal then there exists a maximal pattern P' such that $|L_S(P)| = |L_S(P')|$. We then say that P' *subsumes* P .

Using the above definitions we can succinctly describe the problem addressed by TEIRESIAS as follows:

Problem Definition

Input: A set $S = \{s_1, s_2, \dots, s_n\}$ of sequences from Σ^* , and integers L, W, K where $L \leq W$ and $2 \leq k \leq n$.

Output: All maximal $\langle L, W \rangle$ patterns in \mathcal{C}_{TEIR} with support at least k in the set S .

In what follows, the letters L, W and K are used to denote the parameters specified in the Problem Definition given above and are assumed to have values which have been provided by the user. Furthermore, when the term “pattern” is used without further qualification it will always imply an $\langle L, W \rangle$ pattern.

2.3.2 Hardness

In the Problem Definition given above there is one parameter, namely the variable W , which makes the problem tractable. In the absence of the restriction that this *density* parameter imposes on the pattern language \mathcal{C}_{TEIR} , the problem that TEIRESIAS solves would be NP-hard. In this subsection we explain why.

Consider the following decision problem:

The Longest Common Rigid Pattern (LCRP) Problem

Input: A set $S = \{s_1, s_2, \dots, s_N\}$ of sequences from a discrete alphabet Σ .

Question: Is there a pattern in $P \in \mathcal{C}_{TEIR}$ matched by all the sequences in S ?

We will show that the LCRP problem is NP-complete. The LCRP problem is a restricted version of the *Longest Common Subsequence (LCS)* problem, shown to be NP-complete in [73] (in the case of the LCS, the pattern that one looks for belongs to the pattern language $\Sigma(\Sigma \cup \{*\})^*\Sigma$). The NP-completeness proof given below is obtained by modifying the proof for the LCS problem given in [73].

We use a reduction from the Vertex Cover problem. Consider an undirected graph $G = (V, E)$ where $V = \{1, 2, \dots, n\}$ and $E = \{e_1, e_2, \dots, e_r\}$, where $e_i = \{x_i, y_i\}$ and without loss of generality we can assume that $x_i < y_i$. Given such a graph we construct $(r + 1)$ strings in the following way:

- $s_0 = 1\ 2\ 3\ \dots\ n$.
- For each $1 \leq i \leq r$:

$$s_i = 1\ 2\ \dots\ x_{i-1}\ z_i\ x_{i+1}\ \dots\ y_i\ \dots\ n\ 1\ 2\ \dots\ x_i\ \dots\ y_{i-1}\ z'_i\ y_{i+1}\ \dots\ n$$

where no two z_i, z'_i, z_j, z'_j , $i \neq j$ are the same.

Let $S = \{s_0, s_1, \dots, s_r\}$. All the strings s_i are over the alphabet

$$\Sigma = \{1, 2, \dots, n, z_1, z'_1, z_2, z'_2, \dots, z_r, z'_r\}.$$

The following observation is immediately clear:

Observation: *Any pattern that matches all the strings in S cannot contain any z_i or z'_i character. Furthermore, due to the string s_0 , if two regular characters $1 \leq i < j \leq n$ appear in a pattern matched by all strings in S , then i appears before j in that pattern.*

Additionally,

Claim: *The graph G has a vertex cover of size k if and only if there exists a rigid subsequence with $n - k$ regular characters that matches every string in S .*

Proof: First assume that $V_k = \{i_1, i_2, \dots, i_k\} \subset V$ is a vertex cover for G . Let P be the pattern resulting from s_0 when we replace all characters in V_k by don't-care characters and remove any leading and trailing don't-cares. Let $l \in V - V_k$ be the first character of P . By construction, P matches s_0 . Consider now an arbitrary string s_i , $1 \leq i \leq r$, corresponding to the edge $e_i = \{x_i, y_i\}$. Since V_k is a vertex cover x_i or y_i belong to that set. Without loss of generality, assume that $x_i \in V_k$.

Consider now the substring w_l of s_i that starts at character l , in the first half of s_i and is such that $|w_l| = |P|$. Then, $w_l \in \mathcal{L}(P)$. To see why, observe that the only problem for the match of w_l to the pattern P could be the character z_i . However, because of the way P is constructed and the fact that $x_i \in V_k$, z_i is matched either to a don't-care (which is OK) or falls outside the span of the pattern P (which is also OK).

For the second part of the proof, assume that some pattern P with $(n - k)$ regular characters, namely the characters $V - \{i_1, i_2, \dots, i_k\}$, matches all strings in S . We will show that the set $V_k = \{i_1, i_2, \dots, i_k\}$ is a vertex cover for G .

Again, let l be the first character of P . Consider an arbitrary string $s_i \in S$. The string s_i matches P at the substring starting with the character l at the first or second half of s_i . Without loss of generality, assume that the substring at the first half is the one matching P . Then the character z_i of s_i is either outside the span of P or is matched to a don't-care character (our observation above dictates that z_i cannot be part a regular character in P). In either case, we can conclude that $x_i \in V_k$, i.e. the set V_k covers the edge $e_i \in E$.

□

2.3.3 Implementation

TEIRESIAS begins by scanning the sequences in the input set S and locating all elementary patterns with support at least K . An elementary pattern is just an $\langle L, W \rangle$ pattern containing exactly L residues. Figure 2.5 gives an example of the scanning process for a particular set S of input sequences and particular values for the parameters L and W .

The result of the scanning process described above is a set EP containing all the elementary patterns (and their associated offset lists) which satisfy the minimum support requirement. This set is the input to the convolution phase. To understand how the convolution works, it helps to realize that the scanning phase of the algorithm breaks up all the patterns that exist in the input into smaller pieces. The task, then, of the convolution phase is to put these pieces back together (in a time/space efficient way) in order to recover the original patterns.

The key observation behind the convolution phase is that an original pattern P can be reconstructed by piecing together pairs A, B of intermediate patterns such that a suffix of A is the same as a prefix of B . For example, consider again the set of sequences S given in Figure 2.5. This set contains the pattern “F.ASTS”. It is possible to reconstruct this pattern in the following way (see also Figure 2.6):

- Combine together the elementary patterns “F.AS” and “AST” into the pattern “F.AST” (observe that “AS” is both a suffix of the pattern “F.AS” and a prefix of “AST”).
- Combine the newly generated pattern “F.AST” with the elementary “STS” to get “F.ASTS” (again, “ST” is a suffix of “F.AST” and a prefix of “STS”).

To make the above description more precise, we need the following definitions: given any pattern P with at least L residues, let $prefix(P)$ be the (uniquely defined) subpattern of P that has exactly $(L-1)$ residues and is a prefix of P . For example, if $L = 3$, then

$$prefix("F.ASTS") = "F.A", \quad prefix("AST") = "AS".$$

Similarly, let $suffix(P)$ denote the suffix subpattern of P with exactly $(L-1)$ residues. Again, for $L = 3$,

$$suffix("F.A...S") = "A...S", \quad suffix("ASTS") = "TS".$$

We can now describe a new binary operation, referred to herein as *convolution* (denoted by \oplus), between any pair of patterns: let P, Q be arbitrary patterns with at least L residues each; the convolution of P with Q is a new pattern R defined as follows:

$$R = P \oplus Q = \begin{cases} PQ' & \text{if } suffix(P) = prefix(Q) \\ \emptyset & \text{otherwise} \end{cases}$$

where Q' is a string such that $Q = prefix(Q)Q'$ (i.e. Q' is what remains of Q after the $prefix(Q)$ is thrown away) and \emptyset denotes the empty string. The patterns P, Q are called *convolvable* if $P \oplus Q \neq \emptyset$. Here are some examples for the case $L = 3$:

$$"DF.A.T" \oplus "A.TSE" = "DF.A.TSE", \quad "AS.TF" \oplus "T.FDE" = \emptyset.$$

If two patterns P, Q are convolvable and $R = P \oplus Q$ then the offset list $L_S(R)$ of the resulting pattern R is the subset of $L_S(P)$ defined as

$$L_S(R) = \{(i, j) \in L_S(P) \mid \exists(i, k) \in L_S(Q) \text{ such that } k - j = |P| - |\text{suffix}(P)|\},$$

where $|P|$ denotes the number of characters (counting both residues and don't cares) in the pattern P .

Opting to use convolution as the main tool for reconstructing the original patterns provides a great efficiency advantage over the naive “all-against-all” approach, i.e. examining every possible pair of the intermediate patterns to see if they can be pasted together into a larger pattern. The challenge now is, while using the convolution, to still be able to

- generate all the patterns, and
- manage to quickly identify and discard patterns that are not maximal.

In order to achieve the above goals, two partial orderings on the universe of patterns are introduced. These orderings will be used to guide the way the convolutions are performed. Using them, we can guarantee that (a) all patterns are generated, and (b) a maximal pattern P is generated before any non-maximal pattern subsumed by P . This way a non-maximal pattern can be detected with a minimal effort, just by comparing it against all patterns reported up to that point (comparisons can be made very efficiently using the appropriate hashing scheme to keep track of the maximal patterns). Before giving a precise mathematical definition, we describe these orderings informally.

Let P, Q be two arbitrary patterns. To decide if P is *prefix-wise* less than Q (we write this as $P <_{pf} Q$) we use the following procedure. First, the two patterns are aligned so that their leftmost residues are in the same column. Second, the columns of the alignment are examined starting at the left and proceeding to the right. We stop when we see a column (if any exists) in which one of the aligned characters is a residue and the other is a don't care character. If the residue comes from the pattern P then P is prefix-wise less than Q (see Figure 2.7 for a concrete example).

In exactly the same way we can determine if a pattern P is *suffix-wise* less than Q (we write $P <_{sf} Q$) only that now the alignment is done so that the rightmost residues of the two patterns are aligned together and the examination of the columns starts from the right and proceeds to the left. A formal definition of the two orderings is provided in Figure 2.8.

Figure 2.9 gives pseudo-code describing the convolution phase. The patterns are built inside a stack. When the stack becomes empty, a new elementary pattern P is placed on its top to initiate the next round of convolutions. From that point on and until the stack gets empty again, all the patterns ever placed in the stack will have been obtained from P through successive convolution operations.

The algorithm always works with the pattern T that is found at the top of stack - we call this pattern the *current top*. First, T is extended to the “left” (prefix-wise) by convolving it with all the elementary patterns Q which are such that $Q \oplus P \neq \emptyset$. The convolutions are performed in the order prescribed by the prefix-wise less relation $<_{pf}$, the ties being arbitrarily broken. If the pattern R resulting from such a convolution has support less than K (this can be easily checked by examining the offset list of R) or it is redundant then R is discarded, the current top remains unchanged and the next convolution is tried out. Otherwise, R is placed at the top of stack, thus becoming the new current top and the procedure starts over again, this time with the new current top. After the pattern T at the top of the stack can no longer be extended in the prefix direction, the same process is applied trying now to extend T to the right (suffix-wise).

When extension in both directions has been completed, the current top is popped from the stack. If this pattern is found to be maximal then it is placed in the output set.

2.3.4 Correctness

In order to prove the correctness of the proposed algorithm, we will show that:

- all the maximal patterns are reported, and
- no pattern that is non-maximal is ever reported.

The last of the above two points is also closely related to the performance of the algorithm. Our means of checking the maximality of a patterns is through the function *IsMaximal()* of Figure 2.9. Since this function is called after every successful convolution (i.e. one that generates a pattern R with the prescribed minimum support K) and it requires the lookup of the structure *Maximal* containing the patterns reported up to that point, it is very important to make sure that (i) the number of successful convolutions is not unnecessarily inflated, and (ii) the size of *Maximal* is as small as possible. In order to achieve this goal, the algorithm is so designed as to guarantee that all maximal patterns are reported before any non-maximal pattern. Based on this property we can prove that the structure *Maximal* contains at all times only maximal patterns and that a non-maximal pattern is recognized and discarded immediately.

For the discussion that follows, we introduce some additional terminology. Any $\langle L, W \rangle$ pattern that is the result of a convolution (lines (6), (8) in the pseudo-code of Figure 2.9) is said to be *generated* during the convolution phase. The subset of the generated patterns that make it to the line (9) of Figure 2.9 are called the *reported* patterns. All the patterns generated (reported) between the time an elementary pattern P is placed on the stack and before the next elementary pattern is placed on the stack, are said to generated (reported) *during the expansion* of P .

For every $\langle L, W \rangle$ pattern P define the *core set* of P as the ordered set of elementary patterns $CS(P) = (e_1, e_2, \dots, e_k)$ such that $P = e_1 \oplus e_2 \oplus \dots \oplus e_k$. Let *seed*(P) be that element of $CS(P)$ which is first placed on the stack during the convolution phase. Lemma 1 below is a direct consequence of the definition of maximality.

Lemma 2.3.1 *Let P' be a non-maximal pattern and P the maximal pattern subsuming P' . Then*

$$\text{seed}(P) <_{pf} \text{seed}(P') \quad \text{or} \quad \text{seed}(P) = \text{seed}(P')$$

Lemma 2.3.2 *Let P be an arbitrary pattern. If P is generated at all, then the first time that this happens is during the expansion of *seed*(P). Furthermore, if P is maximal then P is always generated.*

Proof: The proof is an easy induction on the size of the core set $CS(P)$.

□

Theorem 2.3.1 *Let P be a maximal pattern. Then, when P is generated for the first time, no non-maximal pattern P' subsumed by P has been reported.*

Proof: From Lemma 2.3.2 it follows that, since it is the first time that P is generated, the elementary pattern currently under expansion is $seed(P)$. Let now P' be an arbitrary non-maximal pattern subsumed by P . If it is the case that $seed(P) <_{pf} seedP'$ then $seed(P')$ has not yet been expanded (the order of expansion for the elementary patterns obeys the prefix-wise minimum ordering). As a result, P' cannot have been reported since, according to Lemma 2.3.2, it has not even been generated. If, on the other hand, $seed(P) = seed(P')$ then, again by Lemma 2.3.2, P' is generated (if at all) during the expansion of $seed(P)$. Analyzing the pseudo-code in Figure 2.9 reveals that even if P' is indeed generated it has yet not be reported at the time of the generation of P .

□

Given Theorem 2.3.1 above it is easy to see that all the maximal patterns are indeed generated and reported. Furthermore, every maximal pattern is reported before any non-maximal pattern that it subsumes. Given this, we can substantiate our claim that the structure *Maximal* always contains only maximal patterns.

2.3.5 Time Complexity

In the implementation of **TEIRESIAS** the main object of our manipulations is the data structure representing a pattern. This data structure is a pair $(P, L_S(P))$ where P is the string representation of the pattern and $L_S(P)$ is the offset list of the pattern. The list $L_S(P)$ is assumed to be ordered. This means that the offset (x, y) appears before the offset (x', y') if (i) $x < x'$ or (ii) $x = x'$ and $y < y'$. Several other data structures are used, mainly as repositories for patterns. These will be explained as they are encountered.

The remaining part of this section is divided in two parts, one addressing the scanning phase and the other the convolution phase of the algorithm. In what follows we use the character m to denote the size of the input set $S = \{s_1, s_2, \dots, s_n\}$, i.e. $m = \sum_{i=1}^n |s_i|$. Furthermore, we assume that $m \gg W$.

Scanning Phase

During the scanning phase of the algorithm the input is repeatedly examined in order to construct the elementary patterns which will be deposited into the set EP (this set is implemented simply as a linked list of patterns). The components dominating the complexity of this phase are the *for*-loops of lines (2) and (4) in Figure 2.9. The operation of inserting an elementary pattern in the set EP in line (1) takes constant time while the initialization of the array $counts[]$ is an $O(|\Sigma|)$ operation, which can also be considered constant time since our alphabet is fixed. As a result, all that is needed in order to determine the time required by the scanning phase is to compute the number of times the two aforementioned *for*-loops are executed.

In order to facilitate the discussion that follows we introduce the notion of the *template*. Templates provide a conceptual way for constructing the elementary patterns. More specifically, we define an (L, W) template to be an arbitrary string from the set $1, 0$ abiding to the following restrictions.

- it has length between L and W (inclusive),
- it has exactly L '1's,
- it starts and it ends with '1'.

Notice that from an $\langle L, W \rangle$ elementary pattern we can obtain an (L, W) template if we turn every residue to '1' and every don't care character to '0'. Reversely, given an (L, W) template we can produce $\langle L, W \rangle$ patterns by "sliding" it over a sequence (see Figure 2.10): when the template is placed over any sequence position a pattern is obtained by maintaining all residues that are aligned with an '1' and turning into a don't care each residue aligned with a '0'.

Lemma 2.3.3 *For any two integers L, W (with $L \leq W$) let $A(L, W)$ denote the total number of (L, W) templates. Then*

$$A(L, W) = \binom{W-1}{L-1}.$$

The *for*-loops in the lines (2) and (4) of Figure 2.9 force each input character to be examined some number of times. Disregarding any amount of pruning (which occurs

when a pattern $P'\sigma$ in Figure 2.9 is found to have support less than K) each input character is visited no more than

$$A(1, W - L + 1) + A(2, W - L + 2) + \dots + A(L, W) = \sum_{i=1}^L A(i, W - L + i)$$

times (this is actually an overestimate for the characters that are close to the beginning or the end of a sequence). That this claim is true can be verified by inspection of the function *Extend()* used by the code of the scanning phase.

Lemma 2.3.4 *The complexity of the scanning phase is $O(mW^L)$.*

Proof: Each input character is scanned no more than $\sum_{i=1}^L A(i, W - L + i)$ times. Since the input contains m characters, the total time spent in examining them is at most

$$m \sum_{i=1}^L A(i, W - L + i) \leq mL A(L, W) = mL \binom{W-1}{L-1} \leq mLW^{L-1} \leq mW^L$$

□

Convolution Phase

The convolution phase begins with a preprocessing step intended to facilitate the operations to follow. This preprocessing involves the ordering of the set of elementary patterns EP and the generation of the directory structures $DirP$ and $DirS$. Since the complexity of these tasks depends on the size of the set EP , we first give an upper bound for $|EP|$:

Lemma 2.3.5 *The number of elementary patterns discovered during the scanning phase is no larger than $m \binom{W-1}{L-1} / K$.*

Proof: Consider the j -th residue of the i -th input sequence. There exist at most $A(L, W)$ elementary patterns that can start at the offset (i, j) . Consequently, the offset (i, j) appears at most $A(L, W)$ times in the offset lists of all the elementary patterns. Since the input has size m , the aggregate size of all the offset lists in EP is then no larger than $m A(L, W)$. Furthermore, the minimum support requirement dictates that every offset list of an elementary pattern must have at least K offsets. This means that

$$|EP| \leq m A(L, W) / K = m \binom{W-1}{L-1} / K$$

□

Before proceeding with the analysis of the preprocessing step let us give a quick description of the structures $DirP$ and $DirS$. These directory structures are designed for the efficient execution of the lines (5) and (7) in the pseudo-code. Each one of them is implemented as a balanced tree. In the case of $DirP$, every node of the tree is labeled with a string w and contains a non-empty list of all the elementary patterns P with $prefix(P) = w$. The patterns within every such list are ordered according to the $<_{pf}$ relationship. The implementation of $DirS$ is completely analogous, only that here within every node with label w we find all the patterns P satisfying $suffix(P) = w$. Using these structures (and with some minimal bookkeeping inside every pattern) it is possible to execute the lines (5) and (7) of Figure 2.9 in constant time.

Lemma 2.3.6 *The time needed for the preprocessing of the set of elementary patterns EP and the generation of the structures $DirP$ and $DirS$ is $O(W^L m \log m)$.*

Proof: Generating the structure $DirP$ involves (i) ordering the set EP according to the $<_{pf}$ relationship and (ii) for every elementary pattern in EP (considered in the $<_{pf}$ order) locating or creating the balanced tree entry labeled $prefix(P)$ and appending P at the end of the corresponding list. The first of the above steps requires, using any standard sorting algorithm, $O(|EP| \log |EP|)$ comparison, each comparison in our case taking time $O(W)$. The second step takes for each pattern P an (amortized) number of $O(\log |EP|)$ comparisons for locating the node with label $prefix(P)$ and constant time for inserting P into the node's list. Putting everything together and using Lemma 2.3.5 above, we conclude that the time needed for constructing $DirP$ is

$$O(W|EP| \log |EP|) = O\left(W \frac{m}{K} \binom{W-1}{L-1} \log\left(\frac{m}{K} \binom{W-1}{L-1}\right)\right) = O(W^L m \log m).$$

The structure $DirS$ is constructed in an analogous way and the same bounds hold there too.

□

After the preprocessing is done, the main body of the convolution phase commences. The complexity here is dominated by the time spent in the following two operations (the notation used refers to Figure 2.9): (i) convolutions of the pattern T at the top of the stack with appropriate elementary patterns Q , and (ii) checking the resulting patterns

R for maximality. We will address each kind of operation separately, starting with the convolutions.

Lemma 2.3.7 *Let T be an arbitrary pattern at the top of the stack. The number of elementary patterns Q that T is convolved with (lines (6) and (8) of the pseudo-code) is no more than $2|\Sigma|W$.*

Proof: When T is extended to the left in line (6) of Figure 2.9, it has to be combined with an elementary pattern Q such that $prefix(T) = suffix(Q)$. Given that the last $(L-1)$ residues of Q are thus restricted and that an elementary pattern has size no more than W , it follows that there are at most $|\Sigma|W$ elementary patterns satisfying the requirement $prefix(T) = suffix(Q)$. A similar argument holds for the line (8) of the pseudo-code. In total, T is convolved with at most $2|\Sigma|W$ elementary patterns.

□

When the convolution $R = T \oplus Q$ is performed, the offset lists of the patterns T and Q must be compared in order to compute the offset list of the resulting pattern R . Because all offset lists are maintained ordered, this comparison can be done in a single traversal of the two lists, thus requiring $|L_S(T)| + |L_S(Q)| \leq 2m$ operations. Combining this with Lemma 2.3.7 above we get that

Lemma 2.3.8 *For any pattern T that ever appears on top of the stack the total time spent performing convolutions that involve T is $O(Wm)$.*

All that is needed in order to bound the time spent doing convolutions is to determine the number of patterns that are ever placed on top of the stack. As mentioned earlier, the only case that a non-maximal pattern T can be placed on the stack is on the way to building the maximal pattern T' that subsumes T . If we let $rc(T')$ denote the number of residues (i.e. the non-don't cares) in T' then at most $(rc(T') - 1)$ such non-maximal patterns T can be placed at the top of the stack while building T' . Taking into consideration that every maximal pattern is also placed on the stack at some point and using Lemma 2.3.8 we conclude that:

Lemma 2.3.9 *The total time spent carrying out convolutions is*

$$O(Wm \sum_{T' \text{ maximal}} rc(T')).$$

We now turn our attention to the operation of checking the patterns R (the results of the convolutions in Figure 2.9) for maximality. The main data structure used here is the set *Maximal* which at all times contains all the maximal patterns which have been generated up to that point. The set *Maximal* can be thought of as a hash table (although it can also be implemented as a balanced tree for space efficiency). When a new pattern R (with support at least K) is generated, the function *IsMaximal*(R) is called in order to check if *Maximal* contains a pattern which subsumes R . Since this function is called very often, it is important that it be implemented efficiently. Towards that end, every pattern R is assigned a hash value $H(R)$. Ideally, we would like these hash values to have the following properties: (i) no two maximal patterns take the same value, and (ii) for every non-maximal pattern R subsumed by the maximal pattern R' , $H(R) = H(R')$. In such a case and under the light of Theorem 2.3.1 of Section 2.3.4, we conclude that checking a pattern R for maximality is simply a question of computing the hash value $H(R)$ and inspecting the corresponding hash entry: R is maximal if and only if this hash entry is empty.

Unfortunately we can construct reasonable hash functions which satisfy (provably) only the property (ii) above. It is, however, possible to design hash functions which *almost always* satisfy property (i) as well. One such function which has proved particularly appropriate (at least for all test cases we checked it for) maps every pattern R to a pair $H(R) = (|L_S(R)|, \text{diff_sum})$. In order to compute *diff_sum* we first “flatten out” all offsets in $L_S(R) = \{(x_1, y_1), \dots, (x_r, y_r)\}$ by mapping every offset (x, y) to the integer $f(x, y) = y + \sum_{i=1}^{x-1} |s_i|$. The value *diff_sum* is then calculated as

$$\text{diff_sum} = \sum_{i=1}^{r-1} (f(x_{i+1}, y_{i+1}) - f(x_i, y_i)),$$

that is, *diff_sum* is the sum of all the distances between successive matches of the pattern R .

Lemma 2.3.10 *The time required for all the calls to the function IsMaximal() as well as for the insertion of all maximal patterns in the set Maximal (line (9) of the pseudo-code) is*

$$O(W(Cm + t_H) \sum_{P \text{ maximal}} rc(P))$$

where C is the average number of patterns found in a hash entry of *Maximal* and t_H is the time needed for locating the hash entry corresponding to any given hash value.

Proof: The fact that no more than $\sum_P \text{maximal } rc(P)$ are ever placed on the top of the stack, in conjunction with Lemma 2.3.7 leads us to the conclusion that there are at most $2|\Sigma|W \sum_P \text{maximal } rc(P)$ convolution operations performed overall and consequently at most that many patterns ever generated. In the worst case every such pattern R will have to be checked, through a call to *IsMaximal(R)*, against the set *Maximal*. This involves first computing the hash value $H(R)$ (which can actually be done in constant time, or more precisely, can be computed incrementally, as part of the convolution operations leading to the generation of R), second locating the hash entry in *Maximal* corresponding to $H(R)$ (time t_H) and, finally, comparing the offset list of R with the offset list of every pattern found in that hash entry (on the average, time Cm). We note here that if we select a balanced tree implementation for the hash table *Maximal* then the time required in order to locate the hash entry corresponding to any hash value $H(R)$ is $\log F$, where F is the total number of maximal patterns. As a result, $t_H \leq \log F$.

□

Putting now together Lemmas 2.3.4, 2.3.6, 2.3.9 and 2.3.10:

Theorem 2.3.2 *The worst-case time complexity of the TEIRESIAS algorithm is*

$$O(W^L m \log m + W(Cm + t_H) \sum_{P \text{ maximal}} rc(P))$$

Although we have no theoretical bound for the variable C , in all the experiments that we have performed the value of this variable is very close to 1 which indicates that the above described hash function $H()$ will, in the majority of cases, assign distinct values to distinct maximal patterns. .

2.3.6 Experimental Performance

In the course of experimenting with various input sets we have observed that perhaps the single most important factor which affects the performance of the algorithm is the amount of similarity between the input sequences. In order to better evaluate

the relationship between performance and input composition we carried out a number of experiments. The starting point for all of them was a random sequence P of 400 amino acids; the sequence was generated by the random protein generator at <http://www.expasy.ch/sprot/randseq.html> using the amino acid frequencies from the most recent version of SwissProt. We then fixed a percentage X and used the original protein P in order to obtain 20 derivative proteins, each one of them having a pair-wise similarity of about $X\%$ to P . The derivative proteins were obtained through successive applications of the appropriate PAM matrix on P , using a methodology described in [35].

We obtained six such input sets of proteins: for X being 40%, 50%, 60%, 70%, 80% and 90%. For each set we run **TEIRESIAS** using a different minimum support (i.e. value of K) every time. Furthermore, for every choice of the minimum support we used several different values for the parameter W of the algorithm. The parameter L was set to 3; the reason for setting $L = 3$ is that this is the smallest value for which the benefits of convolution become apparent as the prefixes and suffixes used are non-trivial. For each execution of the algorithm we kept track of two things: (a) the running time and, (b) the total number of non-redundant patterns reported by the algorithm. Figures 2.11 and 2.12 provide a graphical representation of the results for the cases $W = 10$ and $W = 15$. The measurements obtained for other values of W are similar.

As expected, the algorithm required just a few seconds when the minimum support was set at about the total number of sequences in the input. Furthermore, the variation in the similarity among the different input sets did not have any observable effect. The reason for this particular behavior is that, no matter what the amount of similarity is, there are not too many patterns that belong to all (or almost all) the input sequences. When we allow for smaller minimum support values, the degree of similarity in the input becomes an important factor. More specifically, the higher the similarity, the longer it takes to produce the final patterns. This is a direct consequence of the fact that as the degree of similarity in the input increases so does the number of distinct patterns that are contained in the input; this observation is typically underestimated by pattern discovery algorithm developers. Examining the curve corresponding to the $X = 90\%$ similarity case one can see that even small decreases in the minimum support create a very large number of additional patterns. So, the increase in the running time is in a

sense anticipated. The important point to make, though, is that there appears to exist a quasi-linear relationship between the execution time of **TEIRESIAS** and the size of the output. As is evident from Figures 2.11 and 2.12 the rate of increase in the running time of the algorithm mirrors almost exactly the corresponding rate of increase in the number of patterns in the output.

From this and additional experiments, we concluded that the size of the output seems to be the main factor affecting the performance of the algorithm: the running time remains reasonably small even for very large input sets if they contain a moderate number of patterns. This is a particularly desirable property since it shows that the amount of work performed by the algorithm is not more than what is absolutely necessary.

2.4 Results

In this section we demonstrate the usefulness of the algorithm by applying it to a number of test cases. We consider two data sets: *core histones* (H3 and H4), and *leghemoglobins*. In the first case, **TEIRESIAS** is used in order to discover similarities across the (weakly related) core histone families H3 and H4. For the leghemoglobins the algorithm is used to extract patterns that are characteristic of the family.

2.4.1 Core Histones H3 and H4

Core histones have been the object of extensive study due to their central role in the packaging of DNA within the cell. These small proteins are rich in positively charged amino acids that help them bind to the negatively charged DNA double helix [114]. The four core histones (H2A, H2B, H3 and H4) bind together into an octameric construct (reminiscent of a cylindrical wedge) that provides the substrate for 146 bps long DNA segments to wrap around, thus creating the nucleosome complexes within the cell chromatin. Examination of this octamer through crystallographic methods [5] has revealed the existence of the *core histone domain*, a particular structural construct found to be shared by all the core histone proteins. As it turns out, this domain plays a vital role in the assembly of histone pairs into dimers which are further combined into the core histone octamer.

Apart from being a deciding factor in the internal architecture of the core histone

octamer, the existence of the domain in all of the core histones has also provided evidence towards the validation of a long standing speculation [85], namely the alleged evolution of the core histones from a common ancestral protein. Recent work [82, 11], based on the manual alignment of core histone proteins along their common domain has indeed established this evolutionary relationship: all histones (as well as the eukaryotic transcription factors CBF-A/C) have been found to contain sequence patterns that are characteristic of the core histone fold [82]. These patterns contain, among other motifs, the pentapeptide KRKT[IV] towards the end of the C-terminal domain [82].

In order to demonstrate its utility, we have used **TEIRESIAS** to analyze the core histone families H3 and H4. Our intention was to see if we could find evidence of the common evolutionary origin of these two families in the form of patterns preserved in both H3 and H4 proteins. This is a challenging task, as there is virtually no observable similarity across the two protein families (while, within family limits, proteins are highly conserved).

For the purpose of carrying out the experiments, we selected representative subsets from both families, making sure to include proteins from a wide range of organisms. We used subsets instead of the entire subfamilies in order to also check the predictive power of the patterns found, i.e. their ability to match ohistone sequences not in the original input set. The input set comprised 20 proteins (13 from the H3 family and 7 from the H4 family) and is shown in Table 2.1.

H33_HUMAN	H32_BOVIN	H32_XENLA	H3_PSAMI	H3_STRPU
H31_HUMAN	H3_ENCAL	H3_CAEEL	H3_PEA	H31_SCHPO
H3_YEAST	H34_CAIMO	H34_MOUSE		
H4_HUMAN	H4_CAEEL	H4_WHEAT	H4_YEAST	H4_SCHPO
H41_TETPY	H42_TETPY			

Table 2.1: The SwissProt labels of the 20 sequences from the H3 and H4 families that were used in the experiments (13 sequences come from the H3 family and 7 from the H4 family).

Processing the input set with **TEIRESIAS** required only a few seconds on an IBM

Power-PC workstation and a large set of patterns was discovered. This set contained patterns that were common to both families as well as patterns that were common to only the members of a single family. In Figure 2.13, we are showing all the discovered patterns (with 4 or more regular characters) which occur in at least 19 out of the 20 sequences (the patterns have been aligned with one representative member of the H3 and H4 families in order to clearly show the patterns' positions in the set). Interestingly enough, there are quite a few patterns appearing in all the proteins in our input set. In order to verify the extent to which these patterns indicate evolutionary relationship (and are not found just by chance) we searched SwissProt (Rel. 34) for sequences matching the most descriptive among these patterns, i.e. those with the largest number of non don't-care positions (Table 2.2).

The 3 patterns we used for the search were very sensitive to the H3 and H4 protein families; they identified almost all the members of these two families. They are also very specific as they generated no *false positives*. In the same table, we are also showing the results of searching the given patterns in the non-redundant database of the core histone sequences maintained at NCBI, the National Center for Biotechnology Information [11, 12]. The high sensitivity of the patterns chosen is clearly demonstrated by that experiment also: the patterns are found to belong in just about all the histone proteins in the non-redundant database.

In Figure 2.14 we are showing a small part of the results produced by TEIRESIAS when applied individually to the H3 (13 sequences) and H4 (7 sequences) subsets. In each case only patterns found to belong in all the members of the respective subset are shown. The figure depicts clearly the extensive degree of amino acid conservation within subfamily boundaries.

Looking at the alignment of the discovered patterns along the *H33_HUMAN* protein, one notices that the following two H3 family patterns

$$P_1 = \text{"A.TKQTA.KST..KAPRKQL.KAA.K.AP..GGVKK.H..P.TVALE.EI.....L"}$$

$$P_2 = \text{"STELLI..PFQRLV.EIAQDFKT.LRFQ..A..ALQE..EA..V.LFEDTNL.AIH.K.V....KD..L.....GER"}$$

fit almost perfectly the two structural domains reported for the H3 family in the PRODOM database, namely the domains with accession numbers 687 (pattern P_1) and 521 (pattern P_2). The first of these two domains appears in 29 sequences all of which are annotated as Histones 3, while the second domain is found in 36 proteins of which 30 are in

Patterns	SwissProt		NCBI	
	H3(33)	H4(20)	H3(81)	H4(59)
G.....T...I.....V..I.....R	27	19	79	58
K.A.....GGVK	24	20	77	59
E.....V...E.....V....K.....G	27	19	76	58

Table 2.2: For each pattern, the number of H3 and H4 members found to contain the pattern are shown. The searches were performed over SwissProt release 34 (containing 33 Histones 3 and 20 Histones 4) and the non-redundant data base at NCBI (containing 81 Histones 3 and 59 Histones 4). It is interesting to note that in searching SwissProt, not a single false positive was generated. Only the pattern “E.....V...E.....V....K.....G” was matched by proteins that were not clearly annotated as histones, namely the proteins *CENA_BOVIN*, *CENA_HUMAN* and *YB21_CAEEL*. All three of these proteins, however, are known to be H3-like proteins: the first two play the role of histones in the assembly of centromeres while the last one is a hypothetical histone 3 protein in chromosome X [9].

the H3 family while the remaining 6 (SwissProt names: *YB21_CAEEL*, *CSE4_YEAST*, *CENA_BOVIN*, *CENA_HUMAN*, *YMH3_CAEEL*, *YL82_CAEEL*) are H3-like proteins.

Verifying Observations

In order to verify the correspondence between patterns and PRODOM domains observed in Figure 2.14 we performed the following experiment: each pattern was slid over all sequences in the SwissProt database. At every offset we counted all characters of the sequence that matched a non-don’t care character of the pattern (this is equivalent to generating a scoring matrix from the pattern where every amino acid position of the patterns contributes a weight of 1 while every don’t care character contributes nothing). The final score assigned to a sequence was the maximum among all the scores of its offsets. As it turns out, even our simple scoring strategy is quite effective; for both patterns the highest scoring sequences (29 for P_1 and 36 for P_2) were exactly those found to be in the corresponding PRODOM domains. Not a single false positive was generated.

2.4.2 Leghemoglobins

Leghemoglobins are plant hemoglobins that show sequence similarity to other members of the larger globin superfamily, especially to myo- and hemo-globins from animals [59]. All globins contain a conserved-histidine signature that represents the heme-binding pocket [59].

BP1_CASGL	HBP2_CASGL	HBPL_PARAD	HBPL_TRETO
LGB1_LUPLU	LGB1_MEDSA	LGB1_MEDTR	LGB1_PEA
LGB1_SOYBN	LGB1_VICFA	LGB2_LUPLU	LGB2_MEDTR
LGB2_SESRO	LGB2_SOYBN	LGB3_MEDSA	LGB3_SESRO
LGB3_SOYBN	LGB4_MEDSA	LGBA_PHAVU	LGBA_SOYBN
LGB_PSOTE	HBP_CANLI		

Table 2.3: The SwissProt labels of the 22 leghemoglobin sequences as they appear in the respective entry of PROSITE.

We presented **TEIRESIAS** with the set of leghemoglobins shown in Table 2.3. This is exactly the set of proteins comprising the plant globin family in the PROSITE (Rel. 13.2) entry with accession number *PS00208*. Our intention was to determine whether the algorithm could identify the heme-binding pocket signature successfully, or any other important pattern.

Figure 2.15 shows those patterns that have been determined by the algorithm to be common to at least 21 of the 22 leghemoglobin sequences of the input set; the patterns are shown aligned with the sequence *LGBA_SOYBN*. Among the patterns belonging to all the input sequences, we find “P.L..HA...F.....A..L...G”. This is essentially the leghemoglobin signature reported in the PROSITE database and describes the heme binding pocket. Another pattern identified by **TEIRESIAS** and appearing in all the input leghemoglobin sequences is “A.L.T.K.....W.....AY..L....K”; furthermore, a search of SwissProt reveals that it is specific to leghemoglobins and creates no false positives. As it turns out, this particular pattern spans the last helix in the structure of the leghemoglobin [59].

Determining Descriptive Power

In order to check the descriptive power of the above two patterns in a greater detail, we used the non-redundant database of proteins at NCBI (including entries from, among others, the SwissProt, EMBL, GenBank and PIR databases) for extracting a big number of leghemoglobin sequences. The sequences were obtained using a standard homology search tool, namely *BLAST* [3, 4] (the algorithm employed by BLAST is discussed in Section 4.2.2), to search for proteins similar to the leghemoglobin *LGBA_SOYBN*; from the search results, only proteins with P value < 0.005 were retained for further processing. The sequences that survived this first screening were then aligned and divided into groups, each group comprising proteins differing in at most two positions. Finally, we constructed a non-redundant set of leghemoglobin sequences by selecting one representative from each group; we assumed that all sequences within a group really code for the same protein and that the small differences are the result of sequencing errors. The result of the process described above was a set containing 60 proteins, the 22 SwissProt proteins listed in Table 2.3 and 38 other leghemoglobins (listed in Table 2.4).

We then proceeded to search the two patterns in this larger leghemoglobin database: both patterns turn out to be very selective (Table 2.5). Each pattern matches every sequence either exactly or within one edit operation (mutation, insertion or deletion), with the exception of one case which requires two mutations. The only sequences missed are those which are fragments and do not contain the region corresponding to the pattern at hand.

On Conserved Regions

The results obtained by *TEIRESIAS* for the leghemoglobin family can be used to highlight another application for the algorithm, namely the derivation of information regarding the conserved regions in a family of sequences. The alignment of patterns shown in Figure 2.15 reveals four such regions, two of them near the N-terminal of *LGBA_SOYBN*, one in the middle of the sequence, corresponding to the heme binding domain, and one close to the C-terminal. These regions are also identified in the *BLOCKS* database as the blocks BL00208A, BL00208B and BL00208C of the plant globin family. The location of these blocks on *LGBA_SOYBN* is shown in Figure 2.15: the correspondence between

OSU76028	G	OSU76029	G	ALFLEGHEMA	G
ALFLEGHEMB	G	SESLBDRLA	G	SESLBDRLB	G
VUU33205	G	VUU33206	G	VFALBA	G
VFALBB	G	VFALBC	G	PHVLBA	G
SOYLBGII	G	GMU4713	G		
S21371	P	S21372	P	S21373	P
S21374	P	S21375	P	S46502	P
S08507	P	S08508	P	S01020	P
S42046	P	GPDRNL	P	GPFJ2	P
GPSYC2	P	A20801	P	A54493	P
VFLBBMR	E	VFLBKMR	E	VFLB1	E
VFLB29MR	E	VFLB49MR	E	MSLEGH12	E
AB004549	O	1102189B	O	711674A	O

Table 2.4: The non-SwissProt proteins included in the non-redundant set of leghemoglobins. Each sequence is represented by its locus and a letter code, indicating the database it came from. The following letter codes are used: **E** (for EMBL), **P** (for PIR), **G** (for GenBank) and **O** (other).

our four regions and the BLOCKS entries is clear. The only difference is that we find two separate areas corresponding to what is reported as the single block BL00208A in the BLOCKS database. We are able to identify the aforementioned regions by assigning a cost to each amino acid position within an input sequence. This cost indicates, roughly, the degree of conservation of that position by taking into account the number of patterns where that position appears preserved. Looking at Figure 2.15, for example, it is clear that the rightmost of the 4 regions shows a heavier concentration of patterns. This is an indication that the amino acids in that region have been preserved relatively well during evolution: in [59] the authors reach the same conclusion by studying the multiple

Pattern	sequences matched exactly	sequences matched within 1 edit operation	sequences matched within 2 edit operations	sequences un- matched due to frag- mented data
P.L..HA...F.....A..L...G	51	1	1	7
A.L.T.K.....W.....AY..L...K	48	9	0	3

Table 2.5: Results of searching the two patterns discovered by TEIRESIAS in the non-redundant leghemoglobin database (containing 60 proteins). The term “edit operation” denotes either a mutation or an insertion or a deletion.

alignment of many globin sequences.

2.5 Discussion

In Sections 2.3 and 2.4 we presented and discussed TEIRESIAS a novel algorithm for the discovery of rigid patterns in unaligned biological sequences. In order to evaluate the utility of the algorithm we have applied it to two distinct sets of inputs. In both cases we demonstrated that the algorithm, in the absence of any context information, is able to derive results of proven biological significance. Furthermore, we briefly discussed how the complete set of patterns that the algorithm generates can be exploited towards the automatic discovery of preserved regions along the proteins of the input set. Finally we described a set of experiments demonstrating the performance and scalability of the algorithm.

We believe that what distinguishes TEIRESIAS from the existing methods of discovering local similarities in biological sequences, is the combined effect of the following two features:

- it is very fast (and scales well with the number and the composition of the patterns in the input);
- it finds *all* the *maximal* patterns with (a user specified) minimum support.

The main reason for the enhanced performance achieved by our algorithm is the utilization of the convolution operation. Most of the existing pattern enumeration algorithms start with a seed pattern and extend it by a single position at a time, checking at every step of the process whether the new pattern has the required support and pruning the search if it does not. This process guarantees completeness of the results but non-maximal patterns are generated and can be very time consuming, especially if the input contains long patterns. Non-maximal patterns show up almost invariably when the minimum support becomes small enough making the performance of algorithms in this class prohibitive at such parameter settings. Consequently, and in order to achieve reasonable running times, either the maximum length of a pattern has to be bounded and/or the minimum support must be set very close to the total number of sequences in the input set.

The convolution operation, on the other hand, permits the extension of a pattern by more than one position at a time, allowing for considerable speed up. Furthermore, our ordering of the intermediate patterns when performing the convolutions gives another performance boost, by avoiding the generation of redundant patterns. The achieved speed gains afford one the ability to look for patterns with very small supports. This is particularly useful when the composition of the input is not uniform, i.e. when it is comprised of sequences that do not necessarily all belong to one group. This was the case with the core histones; although there were a few weak patterns shared by all the proteins, when we reduced the support, larger patterns that distinguished the H3 from the H4 members appeared. In this particular example, the ability to find patterns with small support permitted a finer-grain analysis of the sequences comprising the input. Independent research [16] has already validated the use of **TEIRESIAS** for this kind of analysis.

Another property that differentiates **TEIRESIAS** from existing work, is the kind of structural restriction the user is allowed to impose on the sought patterns. Typically, the speed of the pattern discovery process can be controlled by bounding the length of the reported patterns. This, however, has the drawback that long patterns either escape attention or are broken into multiple redundant and overlapping pieces. In our case, only the parameter W which indicates basically the maximum number of don't care characters between two successive residues in a pattern needs to be set. It thus

becomes possible to discover patterns of arbitrary length (e.g. the case of core histones, Figure 2.14) as long as preserved positions are not more than W residues away.

Finally, TEIRESIAS is guaranteed to report *all* the maximal patterns meeting the structural restrictions set by the user. Other approaches restrict the search space by incorporating a probabilistic or information–theoretic model of importance that is used to decide what patterns to seek. We are of the opinion that the assignment of a measure of importance on the patterns should be disjoint from the discovery process. This way we can guarantee that all the existing patterns are indeed reported. The task of choosing which of them to keep ought to be a post–discovery, problem specific consideration, and should depend on the particular reason the patterns were sought for in the first place. For example, when identifying conserved regions along the input sequences (as in the case of the leghemoglobins) we need the entire set of the existing patterns. On the other hand, when considering issues of pattern specificity/sensitivity then what qualifies as “important” might also be dependent on factors other than the input (e.g. the size and composition of the data base the patterns are to be searched in).

We should point out that in its current implementation the algorithm does not handle flexible gaps. For example, if this functionality were part of the algorithm we would be able to see that the two leghemoglobin patterns

“P.L..HA...F.....A..L...G” and “A.L.T.K.....W.....AY..L...K”

are in fact the two pieces of one larger, flexible pattern; in that larger pattern the two pieces appear separated by a variable number of don’t cares that ranges from 27 to 31 positions.

2.6 Pattern Discovery Revisited

Having defined the problem of pattern discovery in biology and having seen several applications of the discovered patterns, it is instructional at this point to take a step back and attempt to put things into perspective. The discovery of patterns in sets of biological sequences is, essentially, an instance of the *machine learning* problem. In this section we make this connection clear and show how the issues arising in dealing with the discovery of patterns can be formalized and analyzed using concepts from the theory

of machine learning.

The situation we encountered when analyzing the histone and the leghemoglobin families of proteins in Section 2.4 was the following:

Let U denote the universe of all existing proteins, known and unknown. Assume now that $F \subset U$ is a (ideally) well-defined subset U which delineates a protein family, i.e. contains *exactly* those proteins which are members of the family. What we are presented with is a set $S \subseteq F$. The challenge is to find, using S alone, one or more patterns P such that:

- P matches *all* the members of F .
- P matches no member of $\overline{F} = U - F$.

This is a typical machine learning problem: in the *inductive inference* interpretation of machine learning [99] one is given a set of observables (the so called *training set*) and is looking for a *compact description* (or *theory*) which (i) explains the observables and (ii) can predict future phenomena. The parallelism between the two problems is clear: the training set in our case is the set of sequences S , the theory we are looking for is the set of patterns P and the future phenomena are the sequences in the set $U - S$.

So, the pattern discovery problem can be thought of as a restricted version of the following general machine learning problem (also known as the *supervised learning problem using positive examples only*):

The Generic Machine Learning Problem (GMLP)

Input: A set $S \subseteq F$ of elements of a universal set U , where $F \subset U$.

Output: A predicate (also called *theory*) G_{OPT} such that $G_{OPT}(x) = 1, \forall x \in F$
and $G_{OPT}(x) = 0, \forall x \notin F$.

It is quite possible that the predicate G_{OPT} cannot be computed from S alone, either because S is not representative enough of F or because G_{OPT} is uncomputable. In this case, one looks for predicates g that approximate G_{OPT} as well as possible. The ideal evaluation of a proposed predicate g is based on the amount of *false positives* (i.e. members $x \in \overline{F}$ for which $g(x) = 1$) and *false negatives* (i.e. members $x \in F$ for which $g(x) = 0$) that it introduces. Such a predicate is called *specific* for the set F if it creates

no false positives while it is called *sensitive* for F if it generates no false negatives. The specificity and the sensitivity of a proposed solution g of the GMLP can be quantified (following [68]) as:

$$\text{Specificity of } g = F(g) = \frac{|F|}{|F| + f_g^+}$$

$$\text{Sensitivity of } g = N(g) = \frac{|\overline{F}|}{|\overline{F}| + f_g^-}$$

where f_g^+ and f_g^- is the number of false positives and false negatives respectively introduced by g . A predicate g of specificity 1 ($F(g) = 1$) is called a *signature* of F . If g also has sensitivity 1 ($N(g) = 1$) then g is said to be *diagnostic* of F .

Designing an algorithm for the GMLP involves addressing the following issues:

1. Select a class \mathcal{F} of predicate functions to search when presented with a given training set S . The class of predicates defines the *search space* of the algorithm.
2. Devise a methodology (the algorithm) which when presented with a specific training set will produce a set $\mathcal{H} \subseteq \mathcal{F}$ (\mathcal{H} is called the solution space) containing predicates that (hopefully) approximate the predicate G_{OPT} .
3. Rank the proposed solutions in \mathcal{H} . Since a learning algorithm can produce very large solution sets, it is usually beneficial for the end-user to have some means of locating the most relevant (according to some criterion) of the proposed solutions.

Deciding which class \mathcal{F} of predicates to use is a domain-specific issue; it depends on domain knowledge regarding the features/properties that differentiate among members of F and \overline{F} . In the case of biological sequences, these features are regions of conserved residues and one can choose to represent them in a variety of ways (e.g. patterns, profiles, hidden Markov models). The set \mathcal{F} imposes some structure on the chosen representation in an effort to balance two usually conflicting requirements: (i) that the search space be expressive enough to contain signatures for most of the potential inputs, and (ii) that it allow the design of efficient algorithms for the exploitation of the search space. For the pattern discovery problem discussed in this chapter, \mathcal{F} was the particular pattern language used.

Ranking the solution space provides a way of evaluating the predicates in \mathcal{H} . In the discovery problem, every pattern P satisfying the minimum support requirement

introduces into the solution space a predicate g_P defined as:

$$\forall x \in \Sigma^*, g_P(x) = \begin{cases} 1, & \text{if } x \text{ matches } P \\ 0, & \text{otherwise} \end{cases}$$

In analyzing the histone and hemoglobin families in Section 2.4, we saw that there can be a big number of patterns with the requested minimum support. How are these patterns to be evaluated? Ideally, one would use the specificity and sensitivity ($F(g_P)$ and $N(g_P)$) measures. In practice, though, these values can not be computed since the sets F and \overline{F} are not really known. In the text, we have sidestepped this problem by letting F be the set of all the *known* members of a family. The specificity and sensitivity of the resulting patterns were then evaluated in the context of that F . While being sufficient for evaluating the utility of a learning algorithm, this approach is merely an instructional tool and cannot be used in real instances of the GMLP. In such instances the known members of a family are *all* in the training set S . Alternative criteria for evaluating pattern importance are required.

The simplest and most widely used method for ranking the predicates $g \in \mathcal{H}$ is the size of the set

$$R_S(g) = \{x \in S \mid g(x) = 1\}$$

i.e. the subset of the training set recognized by g . This is a sensible strategy, as the least that one expects from g is to identify as many elements of S as possible.

For patterns, more complicated criteria have also been proposed, based either on probability theory [104, 106, 77] or information theory [57]. In the former case, a *pattern probability* is associated with every pattern P , indicating the chance probability that P would have support $|R_S(g_P)|$ in a random input with the composition of S . The smaller the pattern probability of P , the highest its ranking. In the latter case, the pattern is evaluated by combining the amount of information every position of the pattern carries. Single residue positions have the highest information content while don't-care characters and flexible gaps are considered of little value.

Elaborate ranking strategies such as those mentioned above are really useful when evaluating patterns with a relatively small support and/or a rather simple structure (e.g. containing only a small number of residues), i.e. patterns that can *almost* be the result of chance. In the supervised learning problem mentioned above, such a need arises very

rarely. The reason is that the training set S is provided by an expert and it is indeed composed of sequences from the same family. Consequently, the set S will tend to contain patterns of a high support (unless the sequences in the family are very divergent).

Ranking can be a real help mainly when dealing with the *unsupervised learning* variation of the GMLP. In this case, the set S contains samples from more than one families. The task here is to *cluster* the sequences in S according to their respective families and compute signatures for each cluster. This was the situation (in part) with the histone set we examined in Section 2.4. Although all the sequences in the set were histones, there was an extra level of grouping into the H3 and H4 subfamilies. In this case the situation was relatively simple, as there were only two subfamilies and the signature of each subfamily was much stronger than the signature of the family as a whole.

In the next chapter, we are going to look at a complicated situation, where the input contains a lot of subfamilies and where signatures can be *weak*. There we will discuss how to statistically evaluate the discovered patterns.

Overfitting the Training Set

Another issue arising when evaluating the patterns produced by a pattern discovery algorithm is that of the potential *overfitting* of the training set. If the training set comprises of mostly highly similar sequences, then many of the discovered patterns are going to have high support and a rich composition of residues. We have already encountered such a situation in the analysis of the H3 family where the following pattern was discovered:

$$P_2 = \text{“STELLI..PFQRLV.EIAQDFKT.LRFQ..A..ALQE..EA..V.LFEDTNL.AIH.K.V....KD..L.....GER”},$$

From a ranking stand point, this pattern is a perfect specimen: it covers all the H3 members of the training set and it has a high informational content. But it may be *too* specific, i.e. it might fail to be matched by H3 proteins that are evolutionary distant from the members of the subfamily that comprise the training set. Such patterns are said to *overfit* the training set. The danger of overfitting can be alleviated if we allow *approximate* matches of the pattern P . In this case, the predicate g_P introduced by a

pattern P should be redefined as:

$$\forall x \in \Sigma^*, g_P(x) = \begin{cases} 1, & \text{if } x \text{ approximately matches } P \\ 0, & \text{otherwise} \end{cases}$$

This strategy was used in Section 2.4; in the histones example when the pattern P_2 mentioned above was “slid” over the sequences in SwissProt. And in the leghemoglobins when the sequences in the compiled non-redundant data base were matched to a discovered pattern while allowing a small number mutations and insertions/deletions. The issue of overfitting will come up again in Chapter 4, when discussing the search engine proposed there.

s = abcdef

s₁ = Abcdef
s₂ = aBcdef
s₃ = abCdef
s₄ = abcDef
s₅ = abcdEf
s₆ = abcdeF

Q = {s₁, s₄, s₅}

s₁ = Abcdef
s₄ = abcDef
s₅ = abcdEf

P_Q = bc..f

Figure 2.1: The sequences s_1, \dots, s_6 are generated from the sequence s by following the character mutation procedure described in the text. Then, for the set of sequences $Q = \{s_1, s_4, s_5\}$, the pattern P_Q is generated by aligning all the sequences in Q and turning into don't cares all the columns that contain more than one character. Notice that although there are other patterns in \mathcal{C} that are matched by all strings in Q (e.g. the pattern "b...f"), P_Q is the only one that does not match any sequence other than those in Q . In that sense, P_Q is *maximal* for Q .

$$S = \{s_1 = \text{ABFGRP}, s_2 = \text{BDFLRP}, s_3 = \text{AFRP}\}$$

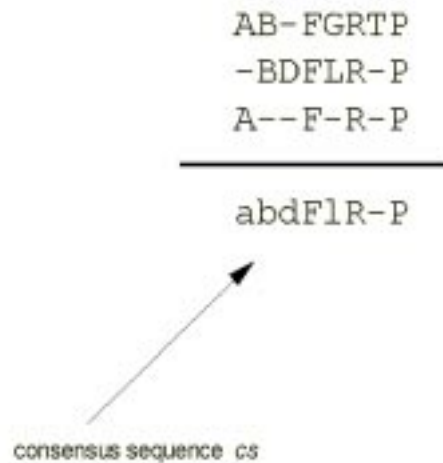


Figure 2.2: Multiple alignment of the sequences s_1, s_2, s_3 . Capital letters in the consensus sequence cs correspond to conserved positions in *all* the input sequences. The set of edit operations that transform an input sequence s_i into the consensus sequence cs can be deduced from the alignment of their respective characters. Assume that x is a character of s_i aligned to a character y of cs . If (i) $x, y \in \Sigma$ then the operation is a mutation from x to y , (ii) $x = '-'$ then the character y is inserted in s_i at the position occupied by x , and (iii) $y = '-'$ then the character x is deleted from s_i .

Protein Name	Part of Protein sequence
H3_CAEEL	ARTKQTARKSTGGKAPRKQLATKAARKSAPASGGVKKPHRYPGTVALREI
H3_ENCAL	ARTKQTARKSTGGKAPRKQLATKAARKSAPATGGVKKPHRFRPGTVALREI
H3_PEA	ARTKQTARKSTGGKAPRKQLATKAARKSAPATGGVKKPHRFRPGTVALREI
H3_PSAMI	ARTKQTARKSTGGKAPRKQLATKAARKSAPATGGVKKPHRYPGTVALREI
H3_STRPU	ARTKQTARKSTGGKAPRKQLATKAARKSAPATGGVKKPHRYPGTVALREI
H3_YEAST	ARTKQTARKSTGGKAPRKQLASKAARKSAPSTGGVKKPHRYPGTVALREI
H4I_TETPY	-----AGGKGG-----KGMGKVG--AKRHSKRSNKASIEGI
H42_TETPY	-----AGGKGG-----KGMGKVG--AKRHSKRSNKASIEGI
H4_CAEEL	-----SGRGKGG-----KGLGKGG--AKRHR-KVLRINIQGI
H4_HUMAN	-----SGRGKGG-----KGLGKGG--AKRHR-KVLRDNIQGI
<pre> artkqtarkstgGkprkqlatkaarKsapatGgvkKpHryrpgtvalreI </pre>	


consensus sequence 

Figure 2.3: Multiple sequence alignment of histone sequences from different organisms. Histones are proteins used for packing the DNA in chromosomes. Here we show members of two distinct histone subfamilies: core histones 3 (proteins named H3_*) and core histones 4 (proteins named H4_*). Regions of these proteins have been aligned and the consensus sequence is shown here. Capital letters in the consensus sequence indicate positions conserved in all proteins. For the other positions we have used lower case and selected for every alignment column the amino acid that appears more often.

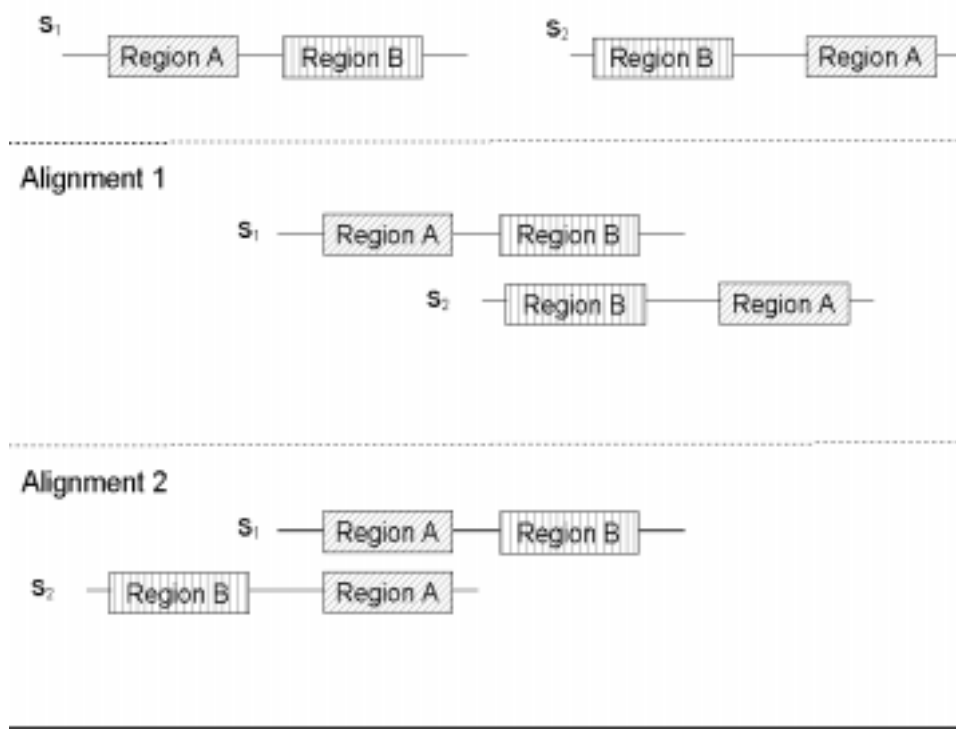


Figure 2.4: The sequences s_1 and s_2 contain two domains, denoted by the patterns A and B . The domains are swapped with respect to one another between the two sequences (this is a common situation in amino acid sequences [43]). This domain–swap forces any global alignment of the two sequences to miss one of the two domains.

$L = 3, W = 4, K = 3$

$S = \{s_1 = \text{"SDFBASTS"}, s_2 = \text{"LFCASTS"}, s_3 = \text{"FDASTSNP"}\}$

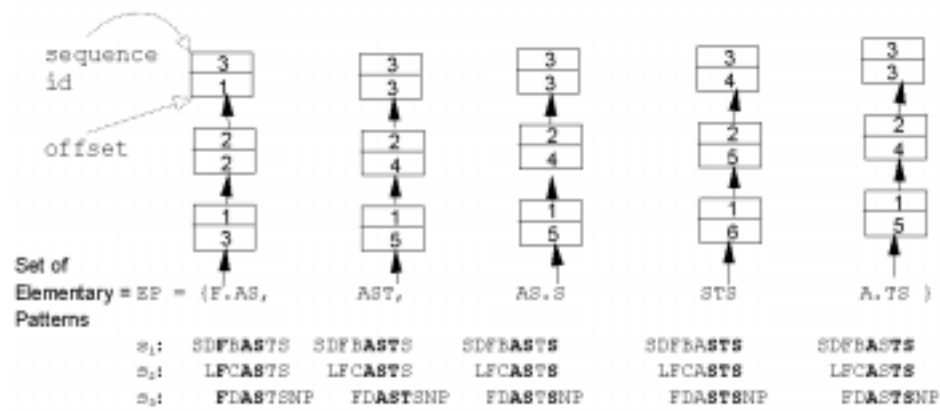


Figure 2.5: The scanning process builds the set EP of all $\langle L, W \rangle$ elementary patterns with support at least K . Each elementary pattern in the set has an associated offset list. For the set S here, the only $\langle 3, 4 \rangle$ patterns that appear in all the three input sequences are the following: “F.AS”, “AST”, “AS.S”, “STS”, “A.TS”.

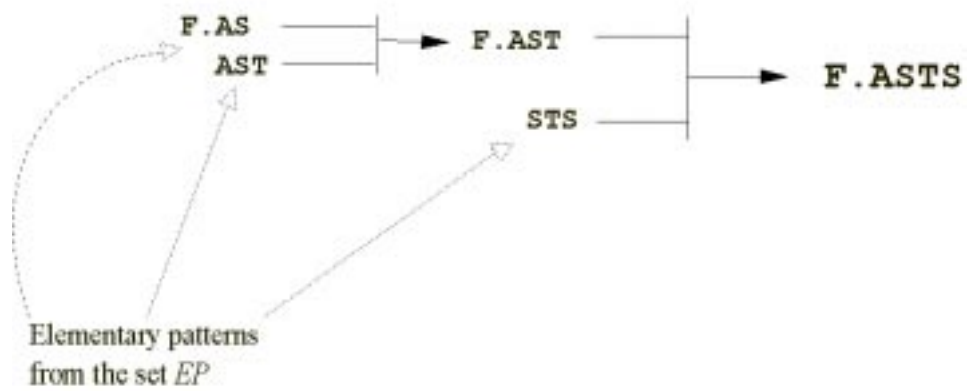


Figure 2.6: Reconstructing the maximal pattern “F.ASTS” from the elementary characterpatterns gathered in the scanning phase.

P = ASD...F
 Q = SE.ERF.DG

Prefix Wise Less

Step 1: Align the patterns along their left-most columns

P: ASD...F
 Q: SE.ERF.DG

Step 2: Locate the first column containing both a don't care and a residue character (this column is marked with an x below).

P: ASD...F
 | | x
 Q: SE.ERF.DG

Step 3: The pattern with the residue character in the x-marked column is the "smallest".

$P <_{\text{pref}} Q$

Suffix Wise Less

Step 1: Align the patterns along their right-most columns

P: ASD...F
 Q: SE.ERF.DG

Step 2: Locate the first column containing both a don't care and a residue character (this column is marked with an x below).

P: ASD...F
 x |
 Q: SE.ERF.DG

Step 3: The pattern with the residue character in the x-marked column is the "smallest".

$Q <_{\text{st}} P$

Figure 2.7: Deciding which of two patterns is prefix-wise (or suffix-wise) less than the other.

Let $\sigma_1, \sigma_2 \in \Sigma$ and $x, y \in (\Sigma \cup \{'.\})^*$. Then:	
<ul style="list-style-type: none"> • $\sigma_1 x <_{pf} \emptyset, \emptyset <_{pf} .x, \sigma_1 x <_{pf} .y$ • $\sigma_1 x <_{pf} \sigma_2 y$ iff $x <_{pf} y$ • $.x <_{pf} .y$ iff $x <_{pf} y$ 	<ul style="list-style-type: none"> • $x\sigma_1 <_{sf} \emptyset, \emptyset <_{sf} x., x\sigma_1 <_{sf} y.$ • $x\sigma_1 <_{sf} y\sigma_2$ iff $x <_{sf} y$ • $x. <_{sf} y.$ iff $x <_{sf} y$

Figure 2.8: The definition of the two partial orderings on the elements of $(\Sigma \cup \{'.\})^*$. For any two strings x, y we say that “ x is prefix-wise less than y ” when $x <_{pf} y$ and that “ x is suffix-wise less than y ” when $x <_{sf} y$. If neither $x <_{pf} y$ nor $y <_{pf} x$, then x, y are said to be *prefix-wise equal* and we then write $x =_{pf} y$ (suffix-wise equality is defined similarly).

<p> $S = \{s_1, s_2, \dots, s_n\}$: set of input sequences $s_i[j]$: the j-th character in the i-th sequence EP: set of elementary patterns $DirP(w)$: subset of EP containing all the elementary patterns starting with w $DirS(w)$: subset of EP containing all the elementary patterns ending in w $counts[i]$: Offset list corresponding to the i-th character in Σ (Σ is arbitrarily ordered). $Maximal$: set of maximal patterns $IsMaximal(R)$: returns 0 if R is subsumed by a pattern in $Maximal$, and 1 otherwise. </p>
--

Scanning Phase

```

EP = null
for all  $\sigma \in \Sigma$ 
  if support( $\sigma$ )  $\geq K$ 
    Extend( $\sigma$ )
  end-if
end-for

Extend(Pattern P)
offset list counts[ $|\Sigma|$ ]

A = # of regular characters in P
if A = L
  Add P to EP (1)
  return
end-if

for i = 0 to (W - |P| - L + A) (2)
  for all  $\sigma \in \Sigma$  (3)
    counts[ $\sigma$ ] = empty
  end-for
  P' = P concatenated with i dots (4)
  for (x, y)  $\in L_S(P)$  (4)
    if (y + |P| + i) < |s_x|
       $\sigma = s_x[y + |P| + i]$ 
      Add (x, y + |P| + i) to counts[ $\sigma$ ]
    end-if
  end-for
end-for

for all  $\sigma \in \Sigma$ 
  if support(P'  $\sigma$ )  $\geq K$  a
    Extend(P'  $\sigma$ )
  end-if
end-for
end-for

```

^asupport(P' σ) is computed from counts[σ].

Convolution Phase

```

Order EP according to prefix-wise less and let
all entries in EP be unmarked.
Build DirS and DirP.
Maximal = empty
Clear stack
while EP not empty
  P = prefix-wise smallest unmarked element in EP
  (ties are resolved arbitrarily)
  mark P
  if IsMaximal(P) then
    push(P)
  else
    continue
  end-if
  while stack not empty
    start:
    T = top of stack
    w = prefix(T)
    U = {Q  $\in DirS(w)$  | Q, T have not
      been convolved yet}
    while U not empty
      Q = minimum element of U
      (according to suffix-wise less) (5)
      R = Q  $\oplus$  T (6)
      if |L_S(R)| = |L_S(T)| then
        pop stack
      end-if
      if support(R)  $\geq K$  and IsMaximal(R)
        push(R)
        goto start
      end-if
    end-while
    w = suffix(T)
    U = {Q  $\in DirP(w)$  | Q, T have not
      been convolved yet}
    while U not empty
      Q = minimum element of U
      (according to prefix-wise less) (7)
      R = T  $\oplus$  Q (8)
      if |L_S(R)| = |L_S(T)| then
        pop stack
      end-if
      if support(R)  $\geq K$  and IsMaximal(R)
        push(R)
        goto start
      end-if
    end-while
    T = pop stack
    Add T in Maximal (9)
    Report T
  end-while
end-while

```

Figure 2.9: Pseudo-code for the scanning and the convolution phases.

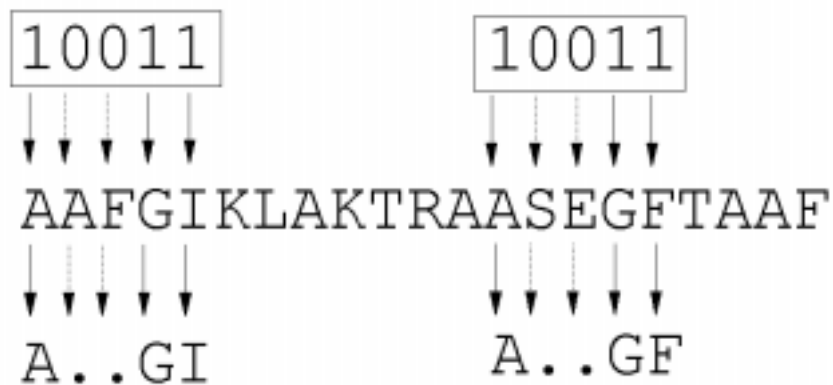


Figure 2.10: The template “10011” placed over the sequence AAFGIKLA KTRAASEGF-TAAF at offsets 1 and 13. The solid lines indicate the alignment of the ‘1’ characters and the residues of the sequence preserved in the resulting patterns. Broken lines point to those residues which are turned to don’t care characters.

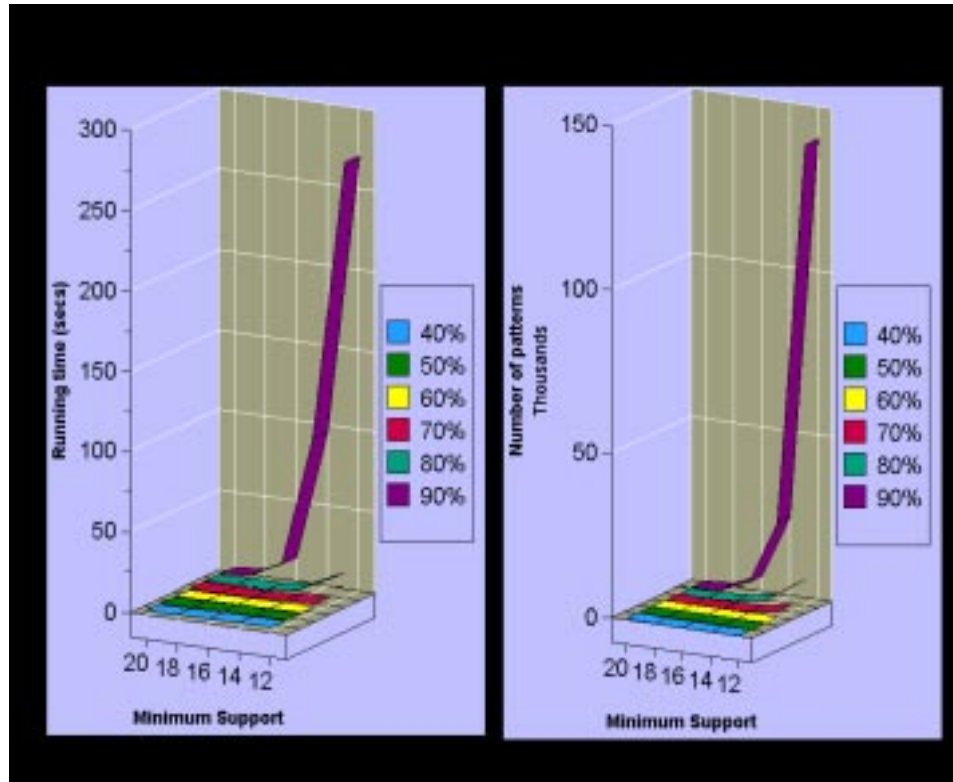


Figure 2.11: For every similarity level (ranging from 40% to 90%) we ran TEIRESIAS on the input set corresponding to that level. For every such input set (containing 20 sequences) we performed a number of experiments each time allowing a different value for the minimum support (the values ranged from 20 down to 12). For each execution of the algorithm we recorded the running time (left graph above) and the total number of patterns reported by TEIRESIAS (right graph). In every run, the value of the parameter W was set to 10.

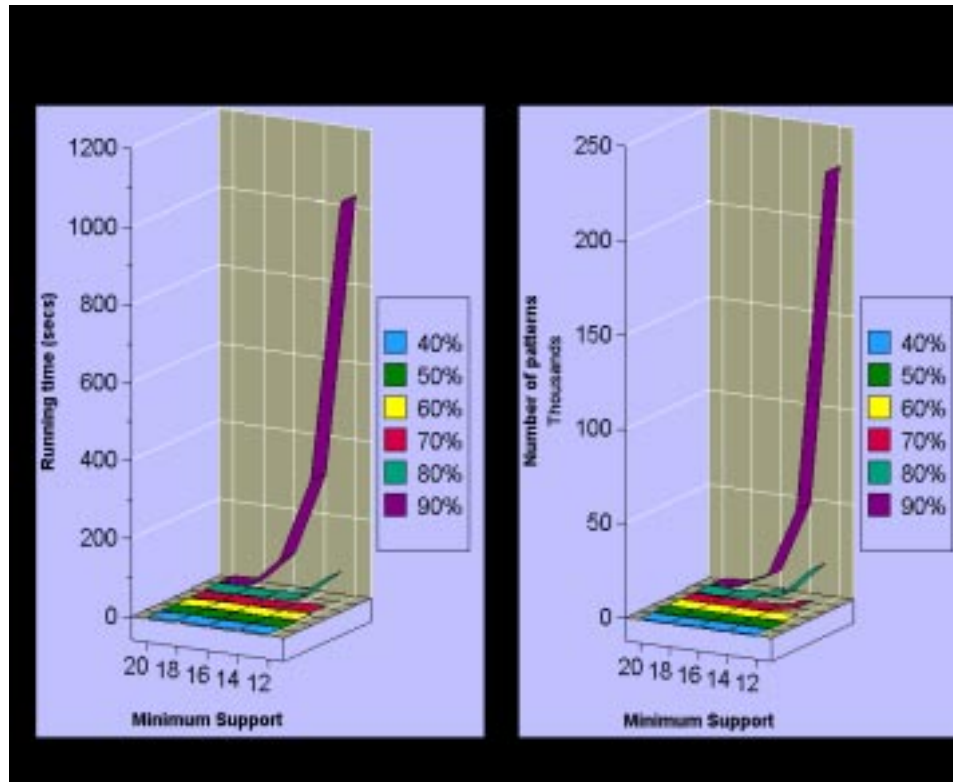


Figure 2.12: Same as Figure 2.11 but with the parameter W taking the value 15.

```

>H33_HUMAN
ARTKQTARKSTGGKAPRQQLATKAARKSAPSTGGVKKPHRYRPGTVVALREIRR YQKSTELLIRKLPFQRLVRELAQDFKTDLRPQSAALGALQEASEAYLVGILFEDTMLCATHAKRVTIMPKDIQLARRIRGERA
-----
Patterns shared by 20/20 sequences
      G.....T.....I.....V..I.....V..I.....E.....V...E.....V.....K...G
K.....R.....V.K
A.RK.....G
      P..R.....L...E V...T.....K.V.....L
      K..L.....E.....V D.....H.....D.....D...G

Patterns shared by 19/20 sequences
      G.....T.....I..L.....V..I.....R
      V.K.....RL.R R.....L.....D.....H..R.....D.....G
      V.K.....A I.K.....RL V...T.....K.V.....L.R..R
      G...KP L.....E.A.....A D.....H...T.....D.....G
      R...T.....K.....K..L.....E.....V D.....HA.....D.....G

```

Figure 2.13: Several of the discovered patterns that are common to both the H3 and H4 families. The patterns are shown aligned with one protein from the H3 family (H33_HUMAN) and one protein from the H4 family (H4_HUMAN) — the latter alignment is shown on the next page. The underlined portion of each protein marks the location of the core histone fold on that protein. Only patterns that occur in at least 19 sequences of the input set’s 20 sequences are shown. It is interesting to observe that for several of the discovered patterns the relative order has not been preserved during evolution.

>H4_HUMAN
 SGRGKGGKGLGKGGAKRHRKVLRDNIQGITKPAIRRLARRGGVKRISGLIYEEITRGVLKVFLNWIIRDAVYTEHAKRKTVTAMDVVYALKRQRTLYGFGG

Patterns shared by 20/20 sequences

```

G.....T...I.....V..I.....R      E.....V..E.....V..K.....G
K.....R.....V..K      V.....T.....K..V.....L
P..R.....L.....E      D.....H.....D.....G
K..A.....GGVK      A..R..K.....G
  
```

Patterns shared by 19/20 sequences

```

G.....T...I..L.....V..I.....R      L.....K.....L.....R
G.....G..KP      L.....E..A.....A
I.....K..R.....Y      L.....K.....AL
AI.....K..I      V.....T.....K..V.....L..R..R
G..K..R.....I      V.....T.....K..VT.....L
K.....LA.....K      D.....HA.....D.....G
L.....R...I.....L      T.....K.....A.....G
R.....L.....D.....H..R.....D.....G
  
```

Figure 2.13: (Continued from last page.) Patterns common to proteins of both the H3 and the H4 core histone families.

(a)

>H33_HUMAN

ARITQTARKSTGGKAPRQQLATKAARKSAPSTGGVKPKRHYRPGTVALREIRR.YQKSTELLIRKLPFQRLVRELAQDFKTDLRPFQSAAGALQEAASEAYLVGLFEDTNLCAATHAKRVTIMP>KD IQLARRIRGERA

PRODOM domain #687

PRODOM domain #521

Patterns shared by all the 13 H3 proteins

A.TKQTA.KST..KAPRQQL..KAA.K.AP..GGVKK.H...P.TVAL.EI.....L A.....STELLI...PFQRLV.EIAQDFKT.LRFQ..A..ALQE..EA..V.LFEDTNL.AIH.K.V....KD..L.....GER

(b)

>H4_HUMAN

SGRGKGGKGLGKGAKRHRKVLRDNIQGITKPAIRRLARRGGVKRISGLIYEEITRGVLKVFLNWRDAVYTYTEHAKRKTVTAMDVVYALKRQRTLYGFGG

Patterns shared by all the 7 H4 proteins

G...G.....G.....I.GITKPAIRRLARRGGVKRIS...Y...R.VLK.FLE.V.RD.VTY.EHA.RKTVT..DVVY.LKRQGR.T.YGFGG
G..G.G.G...R...N...I.....R.GG.KR.....K.....R.....A.....T.....L.....R
GKGGKG.GK.GAKRH.....R...R.G
G...GKG.K.K.....I...A.R...R
G.....R...R.....V.R...E.R.....V
G.....R...R.....V.R...E.R.....V
GG.KR.....K.....R.....T.....L.....R
GG.KR.....K.....R.....T.....L.....R
I.R...G..K.....L.....AV.....A.....L
G..K...R...I.....R.....I.....K.....V
G.....R...V.R.....V.R...E.....V
R.....G..K.....R.....R.....LK
I...A.R...R.....V...V.....V.....G
I...A.R...R.....V...V.....V.....G

Figure 2.14: Discovered patterns that are shared by all of the sequences of the (a) H3, and (b) H4 families. For each family, the patterns are shown aligned with the corresponding human protein. In the *H33_HUMAN* protein, we have also marked the two structural domains of the H3 histone family as they are reported in the PRODOM database ([http:// protein.toulouse.inra.fr /prodom/prodom.html](http://protein.toulouse.inra.fr/prodom/prodom.html)): the first domain (PRODOM #687) covers the underlined part of the sequence while the second domain (PRODOM #521) stretches over the overlined part. As it can be seen in (a) above, the two substantial patterns found for the H3 family fit these two domains almost perfectly.

```

>LGBA_SOYBN
VAFTEKQDALVSSSFEAFKAWIPQYSVVFYTSILEKAPAAKDLFSFLAMGVDPDTPKLTGHAEKLFALVRDSAGQLKASGTVVADAALGSVHAQKAVTDPQFVVKEALLKTIKAAGDKWSELSRAWVEVADELAALKKA
-----
BLO0208A      BLO0208B      BLO0208C
Patterns shared by 22/22 sequences
T.K.....S.E.....Y      T.K.....S.E.....Y
T.K.....S.....Y      T.K.....S.....Y
T.K.....S.E      T.K.....S.E
T..Q.AL      A.L.T.K.W.....AY.L.
      KEA.....W
F...K.....L.
V.K.....L.
A.L.T.....K
Patterns shared by 21/22 sequences
T.Q.AL.....M F.....AP.A...F.L      H.....F.V...A.L.T.K.W.....AY.L.
T.K.....A...Y      E.AP.....F.L      T.K.....A...Y
T.....LV.....K      AP..K..F..L      A.L.T.K.....WS.E...W..AY.L.
T.Q.AL...S      AP.....FS.L      A.L.T.K.....W.....AY.L.
T.Q.ALV S.....AP.....F.L      VVK.....S.....Y.L.
      T.K.....S.....Y.L.
      T.K.....S.E.....L.
      T.K.....S.E.....L
      K.....F.....E.....K      T.K.....S.E.....A
      H.....V...A.L      E...A..AY
      A.....K      A.....D.....K
      F....EA.....A

```

Figure 2.15: Some of the discovered patterns that are common to at least 21 of the 22 sequences in the leghemoglobin input set (shown aligned with the *LGBA_SOYBN* protein). The underlined regions of the protein correspond to the three blocks of the leghemoglobin sequence family, as they are reported in the *BLOCKS* database [44, 46]. The corresponding block names are BL00208A, BL00208B and BL00208C.

Chapter 3

Large Scale Pattern Discovery

Using TEIRESIAS we have explored protein sequence databases and discovered the most frequently occurring sequence patterns. This deterministic identification of patterns has provided us with vistas of the so-called “sequence space”, a much larger, but ill-defined set. The observed patterns, henceforth named *seqlets*, form a finite number of descriptors for this complex space and can be effectively used to describe almost every naturally occurring protein. Seqlets can be considered as building blocks of protein molecules. In many cases they are critical for the function of the proteins they appear in. In other cases they correspond to structural elements of a supporting role. Thus, seqlets can either define conserved family signatures or cut across molecular families and undetected sequence signals deriving from functional convergence. Examples of vicinity in the pattern space are shown both for well known and some interesting newly-discovered cases. This approach delineates the full extent of sequence space and explores the limits of architectural constraints for functional and designed proteins. Pattern discovery results are presented for: (a) a database of six genomes, (b) SwissProt Release 34, and (c) NCBI’s non-redundant database of proteins. The coverage obtained by the discovered seqlets ranges from 74.0% for the six genome database, to 98.3% for NCBI’s non-redundant database. The availability of seqlets that have been derived in such an unsupervised, hierarchical manner is providing new opportunities for tackling a variety of problems which include reliable classification, the correlation of fragments with functional categories, faster engines for homology-searches, and comparative genomic

studies, among others.

3.1 Introduction

Protein sequences define an immensely complex space, with 20^N possible combinations for any sequence of length N . For 100 amino acids, the possible number of protein sequences is $20^{100} \approx 1.27 * 10^{130}$, an astronomical number. Yet, this space is sparsely populated because proteins are related by divergence and form molecular families, which are believed to be evolving by random drift and natural selection processes. The common elements of molecular families are particular conserved positions, often modelled as patterns [52].

Currently, the pattern determination process is based upon the identification of similar proteins, the subsequent generation of multiple alignments and, finally, the selection of the most conserved sites as protein domain signatures [78, 14]. There are two fundamental problems with this approach.

First, due to the lack of appropriate computational tools, it has not been possible to this date to enumerate the most frequent patterns in a large dataset such as a protein database, for example SwissProt; as a result, existing efforts have been confined to the analysis of smaller sets of sequences (mostly containing the members of a single family). Second, most definitions are based on an underlying assumption that patterns can be found only within divergent families [81]. The few convergently-related patterns for functional motifs [29] such as nuclear localization signals are usually not sufficiently specific and cannot be readily discovered by alignment [55].

Ideally, one wishes to carry out pattern discovery in an unsupervised manner. When one subselects a set of proteins from a given database and uses any of the available tools to discover the patterns present in this set, one has made the following two implicit assumptions: first that the members of the formed set are assumed to be indeed related, and second that the member sequences are indeed a single set (as opposed to the union of two or more smaller sets). One would like to discover the most commonly occurring patterns in a sequence database without the above assumptions. Achieving this goal requires, at the very least, the ability to (a) handle very large datasets, and (b) discover all existing patterns. Patterns obtained from datasets of such diverse composition are

expected to unveil previously unobserved protein features both within and across family boundaries, lead to a better understanding of protein architecture, and illuminate the relationships between families that have traditionally been assumed to be unrelated. Additionally, these patterns can be used to derive a natural pattern vocabulary with uses that include automated annotation, reliable classification, sensitive homology derivation, and others.

Up to now, the unsupervised motif discovery has been hampered by the unavailability of pattern discovery techniques that were able to handle data sets of the required size. **TEIRESIAS** however, is not thus constrained. Furthermore, the ability of **TEIRESIAS** to:

- guarantee the discovery of *all* existing patterns with a given minimum support, and
- report only *maximal* patterns

makes the algorithm particularly appropriate for the task at hand.

In this chapter we present the results obtained from applying **TEIRESIAS** on three datasets: the first comprises all the ORFs from 6 complete genomes; the second is the Release 34 of Swiss-Prot, whereas the third is NCBI's non-redundant database of proteins from November 15, 1997. In addition to reporting on and giving a qualitative analysis of the set of discovered patterns we study some of them in more detail; we also describe and discuss potential uses for this set.

3.2 Why Seqlets?

The problem we are addressing here is analogous to the problem of processing an unknown language when the only thing known is the set of its basic units: such basic units can be either syllabic signs (e.g. Linear B, Japanese Kana, etc.) or alphabet letters (e.g. Hebrew, Greek, English, etc.). One attempt to understand such a language would proceed, in the absence of any other information, in a bottom-up manner. From the basic unit we would move to the next level of the hierarchy, that of the vocabulary, then to the level of the syntax. Given the availability of large amounts of sample text in the unknown language, it could be possible to determine not only the vocabulary but the syntax as well, using statistical or combinatorial methods [36]. In fact, it was precisely

this approach that led to the decipherment of Linear B by Michael Ventris in 1953: following the observation that some of the basic units appeared in groups that had the same fixed-size prefix, it was initially suggested that Linear B contained declension. Another observation was that of “...the frequency of certain units and the regularity with which they appeared in a particular context...”. Both of these elements, combined with the assumption that Linear B was a syllabic system, allowed Ventris to build the “syllabic grids” which led to the eventual decipherment of the language [86].

Of course, one would also be interested in determining the semantics for each entry of the vocabulary. However, and unlike the natural language analog given above where deriving semantics is not easy, the same task in the biological context is more plausible: the availability of laboratory analysis methods that can corroborate or refute any hypothesis may facilitate the task of attaching semantics to a word. Finally, it is clear that having access to the semantics of the vocabulary entries and the language’s syntax can in turn allow one to actually use the language and form sensible sentences in it.

Returning to the biological problem that we are examining, the 20 natural amino acids form the alphabet of the unknown language. The seqlets will then correspond to words of this language, and we conjecture that these words are combined together into sentences (and thus viable 3D structures) through a syntax that is imposed on seqlets. Clearly, not all actual syntactic rules are known at the moment. Attaching semantics to the seqlets forming the vocabulary is part of the *functional annotation* task which we will not examine in detail during this discussion.

Recapitulating, we have been given the alphabet of 20 amino acids and a great number of sentences; the task at hand is that of determining as many of the words of the language’s vocabulary as possible. We believe that this task is the first, necessary step towards the comprehensive study of the language used by Nature for building proteins: doing as good a job as possible in determining a complete biological vocabulary is a prerequisite for any future effort that intends to also describe the subsequent levels of the underlying hierarchy (syntax and semantics).

The basic assumption behind our approach is that nature is parsimonious: there exists a finite number of *basic blocks* (i.e. structural and/or functional elements) from which all natural proteins have been built [21, 30]. For any such building block, B , let $CS(B)$ denote all the distinct sequence-fragments which code for it. It is well known

that although similarity at the sequence level implies structural/functional similarity [92], the opposite does not always hold true. As a result, the members of $CS(B)$ can exhibit considerable variability at the sequence level, despite the fact that they code for the same chemical behavior. Clearly, with the approach we are presenting, we can capture only those cases where there still remains sequence similarity among the elements in $CS(B)$; this sequence-level similarity will exhibit itself in the form of seqlets that are shared by the sequence-fragments in $CS(B)$.

Of course, the idea of using sequence descriptors (in the form of patterns, profiles, or HMMs) for representing protein segments of biological importance is not new. Indeed, it has been used successfully in a variety of tools, e.g. PROSITE [9], BLOCKS [46], PFAM [100], PRINTS [6], to name a few. The novelty of our approach is in the methodology employed, the extent of the resulting analysis, and the ramifications of the generated results. To date, sequence descriptors have been obtained by operating on rather small sets of related proteins. Consequently, these descriptors identify regions that play a role in the specific function performed by the sequences of the respective set. In our study, the input set comprised a very large collection of diverse sequences and was processed in an unsupervised manner. Naturally, and with nature's parsimony as a given, one expects to identify not only family-specific functional elements, but also elements of a more elementary and, most importantly, reusable nature: in fact, the algorithm's ability to discover very weak patterns makes it particularly useful for identifying motifs that span multiple families of proteins.

3.2.1 Seqlet Vistas In The Sequence Space

Protein sequences define a complex, multidimensional space which is only a small subset of the theoretically possible sequence space. In essence, this multidimensional space is sparse and the actually encountered proteins occupy only a small subspace of it [71, 67, 38, 101]. A striking demonstration of this statement is the hexapeptide "KPKLGL". To the best of our knowledge this seqlet has not been reported in the literature before, it is very specific for a given family, and accounts for a vast area of sequence space.

In particular, the seqlet "KPKLGL" is very well conserved in the members of the ribulose biphosphate carboxylase (RuBisCo) family. Searching NCBI's Non-redundant database (November 15, 1997) with this hexapeptide retrieves a total of 2646 proteins of

which 2640 are RuBisCo's and 6 are false positives. In order to compare the sensitivity of this pattern with that of chemically neighboring patterns, we have tried replacing the charged amino acid Lysine (K) with Arginine (R), another charged amino acid. Searching the same database with each of the resulting three patterns, namely "K**P**R**L**G**L**", "R**P**K**L**G**L**", "R**P**R**L**G**L**" retrieves 1 (*gi_1171129*), 2 (*gi_1310725* and *gi_2088720*), and 7 (*gi_1708568*, *gi_220141*, *gi_267359*, *gi_267360*, *gi_549365*, *gi_1017788*, and *gi_2454644*) proteins respectively. Clearly, the original seqlet "K**P**K**L**G**L**" is very different than even close neighbors. For comparison, we note that the pattern listed in the PROSITE database entry PS00157, namely the pattern *G*.*[DN]**F*.*K*.*DE*, retrieves 2618 proteins with one of them being a false positive (the notation *[XY]* indicates a position that can be occupied by either of the amino acids *X, Y*).

Given the sensitivity of this seqlet we extracted a region of 16 amino acids around this hexapeptide (5 amino acids + K**P**K**L**G**L** + 5 amino acids). This set of sequence fragments can be thought of as a set of points in a 16-dimensional lattice of 20^{16} vertices. Using multidimensional scaling [65, 66] we progressively projected these points down to three dimensions. As is shown in Figure 3.1, the vast majority of the projections of the oligopeptides aggregated around two attraction points, with 2278 projections in the vicinity of the first attraction point and 261 in the vicinity of the second one; the remaining projections formed two intersecting, almost-planar curves that passed by the attraction points.

Generally, protein sets that share a common motif in many dimensions will look like lines that come very close to one another at a certain neighborhood of the underlying multidimensional space (Figure 3.2 shows a representative such example taken from [15]). In fact, one can think of each seqlet pattern as a compact description of the multidimensional neighborhood that is occupied by the domain that the proteins under consideration share. Clearly, the higher the degree of conservation across the sequences that contain it, the less the spread of the domain's neighborhood. Recalling the definition of patterns handled by **TEIRESIAS** this will translate into a seqlet description of the domain containing fewer don't care characters. As the variation, however, among the domain's instances increases so will the spread of the domain in this high-dimensional space: progressively more don't care characters will be necessary in order to describe the neighborhood. In this discussion, we confine ourselves to domains that can be captured

accurately by the pattern language TEIRESIAS uses.

3.2.2 The Biology of Seqlets

Seqlets are patterns which appear unexpectedly often in a data base of proteins (later on we will define precisely what is meant by “unexpectedly often”). Both in Section 1.2 and throughout this chapter we have provided evidence why biologically crucial stretches of amino acids are expected to have been conserved much better than their less important counterparts. This evidence is the basis of our working hypothesis, namely that patterns that appear often must have “some” biological importance. But exactly how is this statement qualified? What can we say about two sequences that match the same seqlet? Is the seqlet sufficient to describe the function or even the structure of the particular sequence regions matching this seqlet?

Drawing an analog with the natural language problem we described previously, answering these questions amounts to assigning semantics to the words of the language. We mentioned already that assigning this kind of semantics to the discovered seqlets is out of the scope of our work. Although possible, doing so requires extensive physicochemical tests which are both expensive and time consuming. It is, however, possible to get some indication of the potential utility of seqlets by exploiting the existing annotation of known biological sequences. In particular, if a seqlet is matched by protein regions that all share a given functionality then it seems quite reasonable to associate the seqlet with this functionality. Next we are the situations that have been observed. Notice that the seqlets that follow contain *bracketed* expressions; the brackets can be created *after* a seqlet has been obtained, by dereferencing those don't-care characters that exhibit a low variability, i.e. are matched exclusively by only a few, specific amino acids. These amino are then placed within a bracket and this bracket replaces the relevant don't-care character. This way, seqlets become more specific.

1. *one seqlet is specific to a single protein family:* Here, a single family gives rise to a unique seqlet. As an example case, let us consider the family of bacterial histone-like DNA-binding proteins. These are small DNA-binding proteins belonging to two subfamilies: the HU proteins that stabilize DNA from denaturation under extreme environmental conditions and the IHF (integration host factors) with roles

in recombination and transcription initiation [74]. The respective prosite entry, PS00045, lists 35 true positives, 4 partial, 1 false negative and no false positives. The seqlet

$$F[GLT].[FIV]...[RKPQA].[APQES][RST].[GA][RVFH][NK]P.T$$

appears in 36 of them (all the true positives matching the PROSITE pattern plus the one false negative which goes unnoticed); there is no other motif that is shared by these sequences.

2. two or more seqlets are specific to a single protein family: in this case, there are multiple seqlets that are specific to a given family. The family of DEAD box helicases [94] is such an example. The prosite entry PS00039 lists a total of 91 sequences (85 positives, 5 false negatives, and 1 partial). The seqlet

$$[IVLCTA]..[LFVIMK][VI][LMIFV]DE[AS]D.[MLIFC][LFIGMNY]...[FGHLRW]$$

appears in 90 of the prosite's member sequences (i.e. all of the sequences except the partial one, and generates no false positives). A second seqlet appearing in 86 of the member sequences is

$$RG[ILMTV][DHN][IVFL][QPEKSNHDA].[VIL]..[VI][IVFLM][NQLHIS][YFVLI] \\ [DENTHQG][LFIMPVYACT][PASV]...[EDKAQRIHST].[YFLH][VILMQ]H[RT][IVSTAC]G$$

This seqlet also captures *YK04_YEAST* and *YL76_YEAST*, both of which are RNA helicases belonging in this superfamily.

3. one seqlet is specific to more than one protein families: these seqlets correspond to motifs that have been preserved within the members of different families. A typical example of this case is a known structural motif, namely the ATP/GTP binding P-loop

$$G....GK[ST]TL.$$

Detection of the P-loop is not sufficient to classify a sequence into a protein family as there is a wide variety of functionally diverse proteins [93] that exhibit ATP/GTP binding activity. In this case, a seqlet only contains structure information i.e. describes sequence regions that share local 3-dimensional conformation.

In summary, seqlets can describe protein domains (or parts of domains), functional and structural signatures, as well as other sequence characteristics (e.g. traces of common evolutionary origins) not easily amenable to alignment-based analysis. At the end of the Section 3.4 we give a detailed annotation of several frequently occurring seqlets

3.3 Methodology

Unlike the processing of small data sets of related proteins (as were the histone and leghemoglobin families considered in Chapter 2, dealing with large inputs like the protein data bases that we examine here raises a number of technical issues including:

- How should we treat the bias present in a data base (due to the over representation of some families)?
- How should low-complexity protein regions be handled?
- What are the statistics of seqlets?

In this section we address these issues in detail.

3.3.1 Treating Redundancy

Several databases contain groups of highly homologous sequences (e.g. the hemoglobin alpha chain proteins). Such groups not only slow down the pattern discovery process by introducing a huge number of patterns (Sections 2.1 and 2.3.6) but they can also spuriously elevate the significance of a pattern; this happens for patterns that appear many times within a family of highly homologous sequences and only occasionally outside of it.

In order to deal with these problems an input data base D has to be “cleaned up” before the pattern discovery process begins. The cleaning up process that we describe below involves identifying and grouping together highly similar proteins (a technique similar to ours has been independently proposed by [53]).

In particular, we consider the sequences in D in order of decreasing length. At each point we are working with the (currently) longest sequence s_{Long} in D . The sequence

s_{Long} is aligned with every other sequence s currently in D . For the alignment we are using BLAST [3] with the mutation matrix BLOSUM62 [45]: an approximation algorithm like BLAST is sufficient for our purposes as we are only interested in identifying highly similar proteins. If, after aligning the two sequences, s has at least $X\%$ of its positions (in this work we used $X = 50$) identical to corresponding positions of s_{Long} , then s is removed from D and is *associated* with s_{Long} . Finally, s_{Long} itself is removed from D and forms a group $G(s_{Long})$ containing s_{Long} as well as all the sequences associated with it. The sequence s_{Long} is called the *leader* of the group.

Every group among those resulting from the process described above is either (i) a singleton set containing a protein that is dissimilar to all other sequences in the database, or (ii) a group of highly similar proteins with the leader of the group being the longest among them. Groups in this last category (i.e. groups containing at least two members) are called *redundant groups*.

The set D' on which the pattern discovery process will operate is comprised of the leaders of all the groups, redundant and otherwise. After the treatment of D' is finished, each of the redundant groups is separately processed by TEIRESIAS collecting patterns until all the sequences of that group match at least one of the patterns. This approach guarantees that even groups containing multi-domain proteins are treated correctly, by generating at least one pattern per domain (see Figure 3.3). The final set of seqlets is formed by combining the discovered patterns in both D' and the redundant groups. In what follows, the term D' will be used to denote the cleaned-up version of the data base under consideration.

It is worth pointing out that patterns resulting from the processing of the redundant groups will typically be dense (the number of residues is going to be much larger than the number of don't care characters) and long. This is a consequence of the high homology of the group sequences.

3.3.2 Low Complexity Regions

It is known [2] that many proteins contain regions of low complexity, characterized by tandem repeats and/or over-representation of particular amino acids (see the example of Figure 3.4). The existence of such proteins within a database creates a number of problems during the process of pattern discovery because they can give rise to seqlets

which, although statistically important, can be attributed to the compositional bias of the data base.

Several methodologies have been proposed for identifying low complexity regions [91, 116]. Any of them can be used with our approach as well. One should be aware, though, that an extensive masking of such regions can sometimes lead to loss of useful information. Such a situation is exemplified by the pattern “*ALNAA..AA..A*” which was discovered while treating the data base of the six genomes. Although of low complexity, this pattern forms a highly specific signature for proteins involved in chemotaxis. Figure 3.5 shows the proteins from SwissProt Rel. 34 which contain this signature, together with a 23-residue region around the seqlet. It is interesting to note that the seqlet “*ALNAA..AA..A*” allows us to “zoom-in” to a region that exhibits exceptional conservation, and is apparently associated with a functional property common to these proteins.

Other cases of low complexity seqlets with clear-cut functional specificity are also known (e.g. the *leucine-zipper* motif [16]). Examples like these prompted us to adopt a rather moderate approach in masking low complexity regions. In particular, we decided to

- ignore single amino acid stretches (like “FFFFFFFFFFFF”) of length L or more. We do so by replacing all such stretches with random strings of the same length.
- disregard, when computing the offset list of a pattern, matches due to overlapping substrings of a sequence. I.e. if a pattern P is matched twice by a data base sequences s , at the offsets i and j of s and the number $|j - i|$ is less than the length of P (i.e. these are overlapping instances of P within s) then neither i nor j is placed in the offset list of P .

Again, although we have chosen the approach described above for handling low complexity regions, any other technique can be applied as well. The pattern discovery process proposed in this chapter is independent from the particulars of treating these regions.

3.3.3 Seqlet Statistics

Having cleaned up the data base D to be processed and having removed all the low complexity regions as described above, we are ready to start running TEIRESIAS on D' .

Doing so, requires the specification of three parameters: the minimum support K_{min} and the *density* variables L and W . Here we will discuss the methodology used for determining the values of K_{min} , L and W . For the exposition of the material we will use results obtained from our treatment of SwissProt Rel. 34.

As mentioned in Section 2.6, when dealing with large and diverse inputs it is imperative to have some methodology for deciding which patterns are *important*. Most often “importance” is characterized statistically [104, 106, 77] although information–theoretic descriptions have also been used [57]. Here, we will follow the statistical approach. The idea is to focus on those seqlets which are “unexpected” and by virtue of that quality they are also (hopefully) of biological relevance. For our purposes, the significance of a pattern will be described by its support within D' . More specifically, we will seek to define a number K_{min} such that every pattern with support at least K_{min} can be shown to be statistically important. Deciding which minimum support to use is directly dependent on our choice for the parameters L and W . We start by discussing how these density variables are set.

The ratio L/W describes the amount of local homology captured by the discovered patterns. More specifically, consider an $\langle L, W \rangle$ pattern P , two sequences s_1 and s_2 matching P and let A and B be the substrings of s_1 and s_2 respectively that match P , i.e. $A, B \in \mathcal{L}(P)$. Then by the definition of an $\langle L, W \rangle$ pattern the regions A and B have at least L/W of their residues identical. In that respect, the ratio L/W is a lower bound on the homology between any two regions matching a pattern. Consequently, a small L/W will permit the detection of weak similarities. Since several value pairs (L, W) lead to the same ratio L/W what should the exact settings for L and W be? Opting for a large value of L will usually result in a long running time for the pattern discovery process (unless L/W is close to 1). Furthermore, selecting a large L would ignore weak patterns with only a few amino acids. Selecting too small an L on the other hand (e.g. 2 or 3) is useless since in that case the distribution of $\langle L, W \rangle$ patterns with $L + i$ residues (for small i) in the input data base D' is not significantly different from the corresponding distribution in a random database with the amino acid composition of D' .

To make the above point more clear, consider Figure 3.6 which compares the distribution of patterns in the cleaned–up version of SwissProt rel. 34 (for convenience, we

call this version *CleanSP*) with the corresponding random distributions. For CleanSP we computed the support of each $\langle L, W \rangle$ pattern with exactly L residues (for the values of L, W shown in Figure 3.6). Then the results were tabulated creating one row for each possible template (see Section 2.3.5 for the definition of a template): the i -th column of the row corresponding to a given template B indicates the number of patterns (of that template structure) with support i within CleanSP. The random distributions were obtained by following exactly the same approach for 2000 randomly shuffled versions of the entire CleanSP database. In this case the row for a given template B is obtained by averaging the rows corresponding to B in all the 2000 tables. As a result, the i -th column gives a pretty accurate estimate of the mean number of patterns with template B that appear in exactly i sequences within a random data base having the residue composition of CleanSP. In Figure 3.6 we plot the CleanSP results of selected templates against the distribution of the means for the same templates. Although the results presented involve particular templates, there is no qualitative change if other templates are used.

As Figure 3.6 implies, we start distinguishing the compositional bias (in terms of patterns) in CleanSP versus a random data base only when L becomes 5 or larger. In general, the value of L will depend on the size of the underlying data base D' : the larger the data base, the higher this value should be. The results presented in this dissertation have been obtained using the value $L = 6$. For W we chose the value 12, so that the ratio L/W (i.e. the minimum local homology) is 50%.

Having set the values of L and W it remains to decide what the minimum support K_{min} should be. We focus only on patterns with exactly L residues since every larger pattern contains at least one subpattern with exactly that many amino acids. One approach is to select K_{min} so that the probability of a pattern appearing in K_{min} or more distinct sequences is small. Similar significance criteria have been proposed and used before [104, 106, 77]. A closer look at Figure 3.6(d), though, reveals that this approach may be too strict. In particular, consider a support level of $K = 15$. The random distribution indicates that one expects, by chance alone, between one and two patterns at that support level. If the aforementioned criterion were to be used then a pattern with support 15 within CleanSP would be deemed not important. However, the two distributions have a striking difference at that support level. In particular, while the mean of the random distribution at $K = 15$ has a value of about 1.5, there exist about

180 patterns with support 15 within SwissProt .

It seems that if one were to consider the probability of a pattern in isolation the result would be to discard many patterns which, according to the above distribution, are above the level of noise. This observation prompts us to use a different criterion for significance. Instead of looking at individual patterns, we consider together all the patterns of a particular template structure. More specifically, for any given template B and an underlying data base D' let $N_{B,K}$ be:

$N_{B,K}$ = number of patterns with template B which have support K within D' .

Let also $X_{B,K}$ be the random variable (defined over the space of all shuffled versions of D') corresponding to $N_{B,K}$. The minimum support K_{min} is then the first number K for which the following inequality is true:

$$\max_B \{Prob[X_{B,K} \geq N_{B,K}]\} \leq threshold \quad (3.1)$$

where *threshold* is a user-defined probability imposing a level of confidence on the minimum support K_{min} coming out of the above inequality. A smaller threshold leads to a larger value for K_{min} and to a greater statistical importance for the patterns that will be eventually selected.

Since we do not have an analytical description for the distribution of the random variables $X_{B,K}$ we will resort to standard sampling techniques: using the experiments described above with the shuffled version of CleanSP it is possible to compute quite accurate point estimates for both the mean and the deviation of $X_{B,K}$.

More specifically, for any B and K let $m_{B,K}$ and $s_{B,K}$ denote the sample mean and the sample deviation of the random variable $X_{B,K}$: the values of $m_{B,K}$ and $s_{B,K}$ are computed from the 2000 experiments performed on the suffled versions of CleanSP. Let also $\mu_{B,K}$ and $\sigma_{B,K}$ be the actual mean and deviation of $X_{B,K}$. Using elementary statistics [80] we can deduce that with probability at least 0.95 (in the relations below n stands for the number of trials, i.e. $n = 2000$):

$$\sigma_{B,K} \leq \frac{s_{B,K}}{1 + \frac{1.96}{\sqrt{2n}}}$$

$$\mu_{B,K} \leq m_{B,K} + 1.96 \frac{\sigma_{B,K}}{\sqrt{n}}$$

and consequently, with probability no less than $(0.95)^2 \approx 0.9$

$$\mu_{B,K} \leq m_{B,K} + 1.96 \frac{s_{B,K}}{\sqrt{n} \left(1 + \frac{1.96}{\sqrt{2n}}\right)}.$$

Notice that there is no particular reason why we selected to use a confidence level of 95%. Any other value would be applicable as well.

The above inequalities for $\mu_{B,K}$ and $\sigma_{B,K}$ can be used in conjunction with Chebychev's inequality in order to provide upper bounds for the probabilities $Pr[X_{B,K} \geq N_{B,K}]$ used in inequality 3.1. In particular, consider any value $N_{B,K}$ and let C be a constant such as:

$$N_{B,K} = \left(m_{B,K} + 1.96 \frac{s_{B,K}}{\sqrt{n} \left(1 + \frac{1.96}{\sqrt{2n}}\right)}\right) + C \left(\frac{s_{B,K}}{1 + \frac{1.96}{\sqrt{2n}}}\right)$$

Then:

$$\begin{aligned} Pr[X_{B,K} \geq N_{B,K}] &= \\ &= Pr[X_{B,K} \geq \left(m_{B,K} + 1.96 \frac{s_{B,K}}{\sqrt{n} \left(1 + \frac{1.96}{\sqrt{2n}}\right)}\right) + C \left(\frac{s_{B,K}}{1 + \frac{1.96}{\sqrt{2n}}}\right)] \\ &\leq Pr[X_{B,K} \geq \mu_{B,K} + C\sigma_{B,K}] \\ &\leq 1/C^2. \end{aligned}$$

So, using the sample mean and deviation of $X_{B,K}$ we can compute the constant C for the value $N_{B,K}$ at hand. Subsequently, C can be employed to obtain an upper bound for the probability $Pr[X_{B,K} \geq N_{B,K}]$. It is this bound that we use in inequality 3.1. In the results reported here for SwissProt, we chose to use a threshold value of 10^{-11} . The value of K_{min} resulting from 3.1 for these settings turned out to be equal to 15.

To select the threshold, a certain amount of *reverse engineering* has been employed. More specifically, we “conveniently” set the threshold value to be 10^{-11} so that $K_{min} = 15$, i.e. the support level where only 1.5 patterns of a given template structure are expected by chance. There is a tradeoff at play here: we are willing to allow a small number of pattern-induced local homologies which can be the result of chance (the 1.5 patterns above) in order to be able to capture the many more statistically important similarities implied by the other patterns at that same support level present within CleanSP.

A similar procedure as the one described above has been applied in treating the remaining two data bases used in this work, namely NCBI's non-redundant data base

and the data base of the 6 genomes. The resulting values for the parameters L, W and K_{min} are reported in the second column of the Table 3.3.

3.4 Results

In this section, we report on the results we obtain when treating three datasets: (a) a database comprising the sequences from six of the publicly available, completed genomes; (b) the Release 34.0 of the SwissProt database; and, (c) NCBI’s non-redundant database of proteins from November 15, 1997.

Covering the Input

An important quality criterion for the discovered seqlets is their ability to *cover* the sequences in the input set from which they were generated. A sequence is assumed to be “covered” if it matches at least one seqlet. In order to characterize the quality of the seqlet set obtained from a data base D we use the process described below for obtaining a set $U \subseteq D$. Every sequence in U is covered by at least one seqlet. The quality of the coverage will take into account both U and a set P_{kept} of seqlets (the computation of P_{kept} is also discussed below). In what follows, the notation Π_P is used to denote the set of all the sequences from D that match a given seqlet P .

Computation of U, P_{kept}

- initialize both sets P_{kept} and U to the empty set;
- add the most frequent seqlet P_{mf} (i.e. the seqlet matched by the most input sequences, ties broken arbitrarily) to P_{kept} and augment U with the set $\Pi_{P_{mf}}$;
- consider the remaining seqlets in order of decreasing frequency and repeat: if P is the seqlet under consideration, P will be added to P_{kept} and Π_P to U if and only if $|\Pi_P| - |\Pi_P \cap U| \geq d$, i.e. if and only if the seqlet P covers a minimum of d additional sequences from the input set D .

Clearly, the value of d can be a parameter of the analysis. In our discussion below, we are showing results for the values $d = 1, 3, 5$, and 7 .

The procedure just described will be used later on for measuring the quality of coverage in the various test data bases. Ideally, one would prefer that $|U| \approx |D|$ and that $|P_{kept}|$ be as small as possible.

Amino Acid Composition

An additional measure of interest is that of the actual amino acid composition of the seqlets in the set P_{kept} . Given the large number of the involved seqlets and the fact that each seqlet contains a variable number of amino acids, we normalized each seqlet as follows. Let us assume that TEIRESIAS has discovered that the seqlet “G..G.GKST” is matched by 122 input sequences. We begin by creating a 20-counter array where the i -th counter corresponds to the i -th amino acid from the list $A G C D E F Y H I L M V K R N Q P S T W$. We use the number of occurrences (in this example 122) and the seqlet’s composition (in this case 3 instances of G , and 1 instance of K , S and T) to assign values to the twenty counters as shown in Table 3.1. Finally, the counters’ values shown in the table are normalized so that they all sum up to 1. the new resulting array can be plotted by color-coding the values of each counter. An example will be shown later on.

A	G	C	D	E	F	Y	H	I	L
0	3x122	0	0	0	0	0	0	0	0
M	V	K	R	N	Q	P	S	T	W
0	0	1x122	0	0	0	0	1x122	1x122	0

Table 3.1: Amino acid composition of the pattern “G..G.GKST” matched by 122 input sequences. There is one counter for each amino acid and every counter contains the number of times the amino acid appears in the pattern multiplied by the number of sequences (in this case 122) matching the pattern.

3.4.1 The three data bases

The first database comprised the reported sequences from the following six organisms: *Mycoplasma genitalium* [33], *Mycoplasma pneumoniae* [49], *Methanococcus jannaschii* [19], *Escherichia coli* [13], *Synechocystis sp.* [58], and *Haemophilus influenzae* [31]. The second database was the Release 34.0 of SwissProt. The third database was an instance of NCBI’s non-redundant database from November 15, 1997.

In each case we began with the original database, D , and using the methodology of Section 3.3.1 we created the cleaned-up version D' of the original data base along with its associated set of redundant groups. Let G_D denote the *number* of redundant groups generated. In Table 3.2, we are giving details on the sizes of the various sets.

Databases	Sequences / aa in original database	Sequences in redundant groups / Redundant groups	Sequences / aa in cleaned-up database
6 genomes	12,018 / 3,804,214	3,859 / 1,678	9,837 / 3,094,735
SwissProt 34	59,021 / 21,210,388	40,407 / 9,165	27,779 / 10,596,414
NonRedundant	265,536 / 79,287,496	219,180 / 32,455	78,811 / 28,099,451

Table 3.2: For each of the three cases, we are showing the number of sequences and size (in amino acids) of the three databases. Also shown is the number of highly-similar sequences in each case together with the number of groups they form. After removing the redundant groups of highly-similar sequences from the original input, we augment the resulting set by adding the longest sequence from each redundant group. This final product is referred to as the “cleaned-up” database and its size and number of member-sequences is reported as well.

TEIRESIAS was first run on each of the cleaned-up databases and a number of seqlets was discovered. In Table 3.3, we show the parameter choices for each case, the time required to process the respective “cleaned-up” database and the number of discovered seqlets (all processing was carried out on a single, 166 MHz IBM Power-PC processor).

We then sub-selected from the set of discovered seqlets using different values for the

Database	$L/W/K_{min}$	Processing (in hrs)	Number of discovered seqlets
6 genomes	6 / 12 / 10	9.5	33,495
SwissProt 34	6 / 12 / 15	22.5	173,296
Non-redundant	6 / 12 / 20	36	1,326,248

Table 3.3: Parameter choices, single-processor timings, and number of discovered seqlets for each of the three cleaned-up databases.

parameter d . In Table 3.4, we show the surviving number of seqlets for different values of d , and the resulting coverage, for each of the three cleaned-up databases.

For each of the three data bases D , the corresponding set of seqlets was augmented with the results of running TEIRESIAS on each of the G_D redundant groups generated for D . These seqlets were added to those that were discovered by processing the corresponding cleaned-up database, and the coverage of the original database was finally computed. These figures are included on Table 3.4 as well. Also of interest is the graph that shows how the input coverage increases as more and more seqlets get added to the set P_{kept} . Figures 3.7, 3.8, 3.9 graphically depict this for each of the three data bases and for $d = 1, 3, 5, 7$.

It is interesting to note the difference between the genomic database and the two curated ones. A fixed number of seqlets can cover many more sequences in the cleaned-up instances of the curated databases than in the genomic one. For example: a little over 500 seqlets can cover approximately 3,500 sequences of the genomic input but almost 8,000 sequences of the cleaned-up Swiss-Prot. Also, almost 4,000 seqlets can cover approximately 6,700 genomic input sequences and somewhere between 30,000 and 41,500 sequences from the non-redundant database. In other words, a given seqlet achieves greater degree of coverage as the experimental bias of the target database increases. This is not unexpected as in over-represented families one pattern might cover all or most of the family members. On the other hand, as soon as such families are covered by selected patterns, the degree of coverage imposed by the patterns in P_{kept} drops to lower levels. This last observation is made apparent by the plots of Figures 3.8 and 3.9: there

is a certain degree of saturation that manifests itself in the SwissProt and non-redundant database plots; the plot for the genomic database on the other hand exhibits a constant slope through the maximum achieved coverage.

Let us concentrate on NCBI's non-redundant database. In Figure 3.10, we show a pseudo-colored matrix for the amino acid composition of the first 100 seqlets corresponding to P_{kept} when $d = 7$. As is evident from this Figure, there is a clear preference for certain amino acid composition despite the fact that order information within each seqlet has not been preserved; the remaining seqlets of P_{kept} for $d = 7$ exhibit a similar composition bias. And the same holds true for the seqlets of P_{kept} for the other values of d . Analogous figures for the database of the 6 genomes and for SwissProt Rel. 34 have also been produced.

The accompanying set of seqlets that correspond to Figure 3.10 are listed on Table 3.5. The seqlets are grouped in categories depending on their nature: a given category, e.g. the category of the P-loop variations includes several seqlets in order of decreasing number of occurrences; next to each reported seqlet its rank (with '1' corresponding to the most frequently appearing seqlet) is also shown. Recall that in general as the frequency of the seqlet decreases the seqlet becomes increasingly more specific. Wherever this was possible, the seqlets of a category have been aligned. Notice the presence of brackets in some of these seqlets: the lower case letters denote such bracketed expression and are described in the Table's caption.

3.5 Seqlets and Clustering

Many data-rich disciplines resort sooner or later to techniques for *organizing* their data into *semantically meaningful* subcategories. This becomes necessary not only for practical reasons, e.g. for enabling a systematic navigation through a vast collection of information, but also as a means to impose structure and further exploit this structure for the generation of new knowledge. Molecular biology is no exception to this rule, especially with the ever increasing influx of biosequences. Here, the groups of related sequences are called *clusters* and the process followed for filling each cluster is referred to as *clustering*.

A simplified way to think of a cluster is as a set of proteins C along with a *label* that describes the *semantics* of C , i.e. the particular property associated with all the

sequences in the cluster. In most of the clusterings that we are aware of, the semantics are of a *functional* nature: they try to describe the functionality of the members of a cluster. A typical example of a *curated* clustering of this type is the PROSITE data base. A PROSITE entry is labeled with a pattern matched by all the sequences associated with this entry. The sequence regions matching the pattern are all known to play the same functional role. Clusterings based on structure semantics have also been proposed [75] but since the available protein structures are fewer than the known protein sequences, the impact of such efforts has been limited. To appreciate why a clustering can be a valuable tool for biology consider the following idealized situation:

- there exists a clustering $\{(C_1, L_1), (C_2, L_2), \dots, (C_m, L_m)\}$ of all the proteins where C_i is a cluster and L_i is the label of C_i .
- for every $i, 1 \leq i \leq m$, and every protein s the following predicate is computable:

$$F_i(s) = \begin{cases} \text{TRUE}, & \text{if } s \in C_i \\ \text{FALSE}, & \text{if } s \notin C_i \end{cases}$$

If such a clustering did indeed exist then every new protein could be characterized (according to the semantics of the clustering) by using all the labels L_i for which $F_i(s)$ evaluates to TRUE.

For this reason, clustering of molecular data has always been recognized as a pressing need. Early clustering efforts have been *curated* i.e. prepared and maintained by experts. As such, they are extremely valuable since they incorporate domain knowledge. Unfortunately, curation requires manual inspection and, consequently, cannot keep up with the continuously increasing volume of experimental data. As a result, automatic clustering methodologies have been proposed [37, 42, 105, 118, 63, 71].

Automatic clustering techniques resemble the approach that we have described here in that they try to extract information from the processing of large sets of diverse proteins. The goal of automatic clustering is to build *functionally semantic* clusters by only looking at the primary structure of proteins. In most of the cases, one wants to create one cluster per *domain* (see the Biology Primer in Chapter 1 for the definition of domains). This objective cannot be achieved easily: we have already mentioned that functionally related polypeptides have some times quite diverse sequences and this is a hard biological fact. Consequently, one has to settle for the best clustering attainable.

In all of the clustering approaches that we are aware of, the underlying theme is the following:

- For every pair of proteins x, y quantify the potential that they share a domain. This quantification is usually performed by optimally aligning related regions of the proteins and assigning a cost $d(x, y)$ to the alignment.
- Consider a graph whose vertices are all the proteins in the underlying data base and where the edge between the vertices x, y is weighted with the value $d(x, y)$. Employ a clustering technique to cluster the vertices of the graph using the edge weights.

There are several issues arising here. First, *optimally* aligning every pair of proteins is computationally intensive and quickly becomes impractical as the size of the input data base increases. As a result, the alignments are actually performed using approximation algorithms like BLAST. Consequently, weak similarities can remain unnoticed. Furthermore, since fast alignment algorithms tend to disregard insertions/deletions it is possible that a functional domain shared by two proteins x, y appears as two or more local alignments, separated by gaps. In order for the $d(x, y)$ metric to correctly mirror the relationship between x and y , the fragmented alignment has to be repaired [42]. Second, clustering the vertices of a weighted graph necessitates the choice of thresholds that decide when two vertices should be placed in the same cluster. In general, choices of thresholds are more or less arbitrary and are usually made according to the type of clustering that one pursues: high thresholds will allow the detection of only extensive homologies, whereas low thresholds can detect distant relationships. The problem with the latter case is that there is no way of knowing beforehand at what threshold level a “weak” similarity becomes “biologically irrelevant”. Figure 3.11 illustrates this point graphically with a dataset of 16 items. Depending on the selection of thresholds the correct number of clusters can be 2, 3, 4 or 16.

A third problem arises because of the multidomain nature of many proteins. Consider again the Figure 3.3. The leader sequence shown there contains the two domains A and B . Since the other four proteins share extensive similarity with the leader sequence, it is possible for all five proteins to be placed in the same cluster. This, however, would

be wrong as A and B are disjoint domains and should really define distinct clusters. So, special care must be taken in handling such multiple domains.

Clustering entire databases (even with the use of approximation algorithms) is a very time consuming task. Up to now the argument was that speed is not a crucial factor as clustering of a database is performed infrequently (usually once for every new release of the database). These days, however, many new sequences get added to (mostly privately owned) data bases every day, so that clustering occurs much more often and speed is important. Finally, it is important to notice that automatic clustering does not produce labels for the clusters: assignment of the semantics is still a manual process.

Seqlets naturally induce a clustering of the proteins space in the same way that PROSITE patterns do: every seqlet P defines a cluster C_P containing all the sequences matching P . Furthermore, the second property of an ideal clustering, namely the computability of the predicate

$$F_P(s) = \text{TRUE}, \quad \text{iff } s \in C_P,$$

holds as well (we will explore this issue in detail in the next chapter). The main characteristic of this seqlet-induced clustering is its *redundancy*. I.e. there may be more than one seqlets that describe basically the same functional or structural domain. An example of this situation is the following pair of seqlets: $P_1 = "G..G.GKST"$ and $P_2 = "G..G.GKST...L"$. Both of them are variations of the P-loop motif (see Table 3.5). It is not hard to realize that every region matching the second seqlet will also match the first one. In other words, the offset list of P_2 is going to be a subset of the offset list of P_1 . So does it make sense to have separate clusters for P_1 and P_2 ? Or are they really describing the same domain?

The answer to this question is not straightforward. In the example given above it so happens that the two seqlets really describe the same motif. But it might have been the case that the longest of the two patterns represented a multiple domain and the first pattern was just one part of that domain. In that case, the two seqlets would correctly model two distinct clusters.

There seems to be no one "correct" algorithmic way to address the question of merging (or not) seqlet-induced clusters. It is only biological information (namely, the function/structure of the regions described by the seqlets) that can provide a definite answer.

This is the reason why we chose not to provide a specific clustering here. This, however, does not mean that one cannot or should not try to use the seqlets in order to extract more comprehensive clusters of proteins. Any set of criteria can be used for that purpose. The more biologically sound these criteria are, the more credible the resulting clustering. For example, for the two seqlets mentioned above one can assert that if the size of their respective offset lists are comparable, then they probably correspond to the same domain. In what follows, we provide a graph theoretic framework general enough to allow expressing a wide class of such criteria. For the remaining of this discussion, the notation Π_P is used to denote the set of sequences (from the input set) that match the seqlet P . We will also be comparing the offset lists of seqlets. In this setting, an offset $(x, y) \in L_D(P_1)$ will be considered to be the *same* as the offset $(x, y') \in L_D(P_2)$ if either (i) $y = y'$, or (ii) $y' \geq y$ and $y' - y \geq |P_1|$.

Given an input dataset D and the set of seqlets Π associated with D we can define a directed graph $G = (V, E)$, where $V = \Pi$. This graph will be called the *seqlet graph*. The set of edges E can be defined in a variety of ways, each way describing an approach for collapsing multiple seqlet-clusters into one. Figure 3.12 graphically shows a seqlet graph for a small dataset.

In the simplest case, two vertices u_{P_1} and u_{P_2} (corresponding to the seqlets $P_1, P_2 \in \Pi$) are connected with an undirected edge $u_{P_1} \leftrightarrow u_{P_2} \in E$ if and only if $L_D(P_1) \subset L_D(P_2)$. This edge assignment corresponds to a strategy which assumes that two seqlets describe the same functional domain whenever the appearances of one subsume the appearances of the other. This approach is applicable to seqlet clusters such as those described in the example above. Large clusters can then be formed by finding the connected components of the seqlet graph.

More sophisticated cluster relationships are also possible. Consider, for example, the following situation which arised in our experiments. The family of ATP-binding proteins contains the pair of seqlets $P_1 = \text{“LDEPT.L”}$ and $P_2 = \text{“L.DEPT.LD”}$. In Release 34 of SwissProt, there exist 76 proteins in the set Π_{P_1} and 74 proteins in the set Π_{P_2} . However, the set $\Pi_{P_1} \cup \Pi_{P_2}$ contains 108 proteins; clearly, Π_{P_1} and Π_{P_2} are distinct sets with a substantial non-zero intersection. Using the edge set described previously, there would be no edge connecting u_{P_1} and u_{P_2} . However, it may be beneficial to be able to represent such intersecting seqlets in this graph.

In order to capture this requirement, the edge set definition must be redefined as follows: a directed edge $u_{P_1} \rightarrow u_{P_2} \in E$ will connect two vertices u_{P_1} and u_{P_2} if and only if (i) $|L_D(P_1)| \leq |L_D(P_2)|$ and (ii) $L_D(P_1) \cap L_D(P_2) \neq \emptyset$. This definition will correctly introduce an edge between the vertices of the seqlets in ATP-binding proteins. However, it gives no measure of the extent of the implied intersection. Ideally, one would wish to distinguish between the two cases shown in Figure 3.13.

The obvious extension of this first variation is then: a directed edge $u_{P_1} \rightarrow u_{P_2} \in E$ will connect two vertices u_{P_1} and u_{P_2} if and only if:

- $|L_D(P_1)| \leq |L_D(P_2)|$,
- $L_D(P_1) \cap L_D(P_2) \neq \emptyset$,
- $|L_D(P_1) \cap L_D(P_2)| \geq \alpha |L_D(P_1)|$, where $\alpha \in [0, 1]$.

I.e. not only should the offset lists of the two seqlets intersect but the intersection should be a predefined fraction of the smaller offset list. Finally, depending on which graph traversal/processing algorithm one intends to apply to the seqlet graph for extracting clusters, it may be useful to assign weights on the graph edges. For example, each edge $u_{P_1} \rightarrow u_{P_2}$ can be labeled with the cardinality of the intersection of the offset lists of $L_D(P_1)$ and $L_D(P_2)$.

In conclusion, the seqlet graph provides a convenient representation for the application of a large number of clustering approaches. Clustering strategies can be translated in appropriate edge definitions, while the processing task of forming the clusters can utilize a large number of existing graph algorithms.

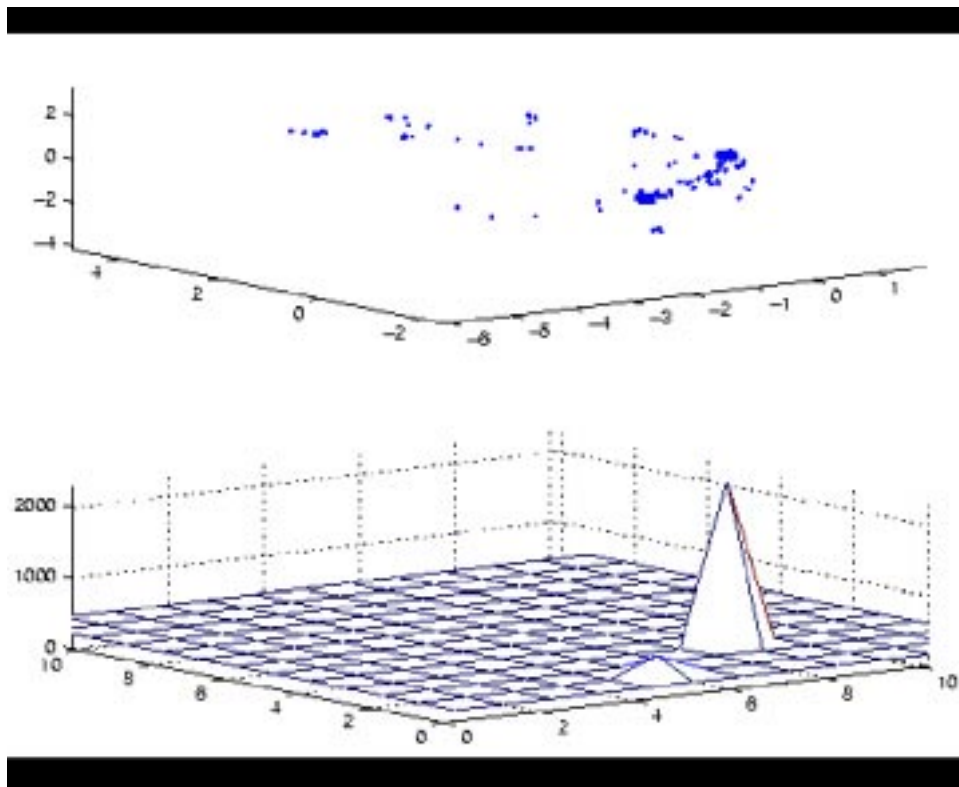


Figure 3.1: The result of multidimensional scaling of the 23 amino acid region centered around the hexapeptide “KPKLGL”. The three-dimensional projections are shown at the top of the figure, whereas the bottom shows a histogram of the cardinality of the points in the neighborhood of the attractors.

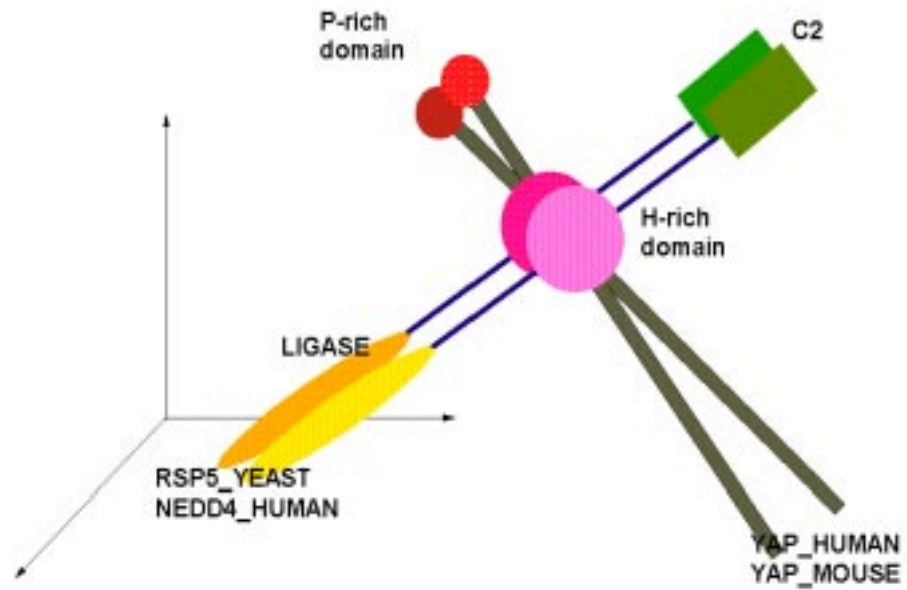


Figure 3.2: An example of proteins sharing domains; the figure is derived from [15]. One pair of sequences (*YAP_HUMAN*, *YAP_MOUSE*) share a Proline-rich domain. The other pair (*RSP5_YEAST*, *NEDD4_HUMAN*) have in common a *ligase* and a *C2* domain. Finally, a Histidine-rich domain belongs to all 4 proteins.

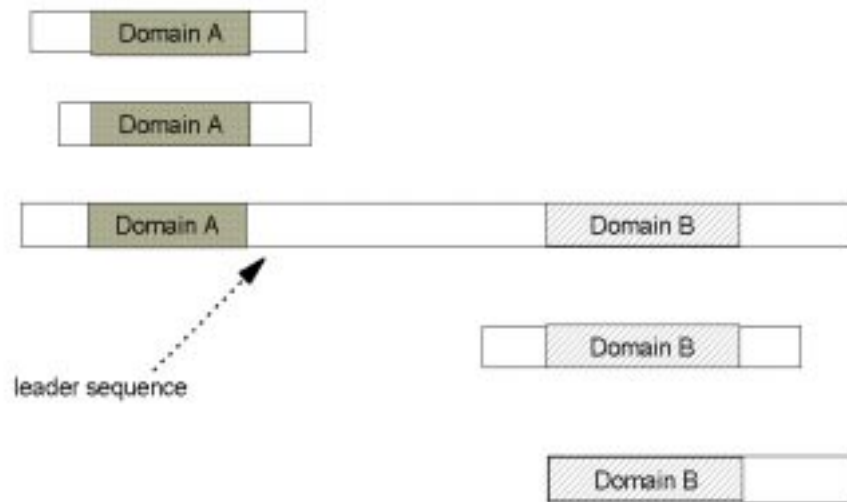


Figure 3.3: Multi-domain groups are generated by leader sequences that contain more than one domains. In the case depicted here, the *leader sequence* will attract the two pairs of the shorter sequences in its group. In order for redundant groups of this sort to be treated correctly (i.e. no useful information be lost), it is important that patterns corresponding to both domains *A* and *B* are generated.

```

Swiss Prot Id:  PAR1_TRYBB
MAPRSLYLLAVLLFSANLFAGVGFAAAAEGPEDKGLTKGGKKGKGEKGTKVGADDTNGTDPD
PEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPEPE
PGAATLKSVALPFAIAAAALVAAF

```

Figure 3.4: The protein shown above is an antigen from the organism *trypanosoma brucei brucei*. It contains a low complexity region where the dipeptide “PE” is tandemly repeated.

Protein name with beginning and end of region	Protein region	Description of Protein function
>AER_ECOLI_364_385	QTNILALNAAVEAARAGEQGKG	AEROTAXIS RECEPTOR
>CPS_CLOTM_344_365	QTNILALNAAVEAARAGQHGKG	PUTATIVE SENSORY TRANSDUCER
>DCRA_DESVH_511_532	QTNLLALNAAIEAARAGDAGRG	CHEMORECEPTOR PROTEIN A.
>HLYB_VIBCH_391_412	QTNLLALNAAIEAARAGEQGRG	HEMOLYSIN SECRETION PROTEIN
>MCP1_ECOLI_374_395	QTNILALNAAVEAARAGEQGRG	METHYL-ACCEPTING CHEMOTAXIS
>MCP2_ECOLI_372_393	QTNILALNAAVEAARAGEQGRG	METHYL-ACCEPTING CHEMOTAXIS
>MCP2_SALTY_372_393	QTNILALNAAVEAARAGEQGRG	METHYL-ACCEPTING CHEMOTAXIS
>MCP3_ECOLI_382_403	QTNILALNAAVEAARAGEQGRG	METHYL-ACCEPTING CHEMOTAXIS
>MCP4_ECOLI_370_391	QTNILALNAAVEAARAGEQGRG	METHYL-ACCEPTING CHEMOTAXIS
>MCPA_BACSU_489_510	QTNLLALNAAIEAARAGEYGRG	METHYL-ACCEPTING CHEMOTAXIS
>MCPB_BACSU_490_511	QTNLLALNAAIEAARAGESGRG	METHYL-ACCEPTING CHEMOTAXIS
>MCPC_SALTY_373_394	QTNILALNAAVEAARAGEQGRG	METHYL-ACCEPTING CHEMOTAXIS
>MCPD_ENTAE_367_388	QTNILALNAAVEAARAGEQGRG	METHYL-ACCEPTING CHEMOTAXIS
>MCPS_ENTAE_376_397	QTNILALNAAVEAARAGEQGRA	METHYL-ACCEPTING CHEMOTAXIS
>TCPI_VIBCH_459_480	QTNLLALNAAIEAARAGEQGRG	TOXIN CORREGULATED PILUS
>TLPA_BACSU_489_510	QTNLLALNAAIEAARAGEYGRG	METHYL-ACCEPTING CHEMOTAXIS
>TLPB_BACSU_489_510	QTNLLALNAAIEAARAGEYGRG	METHYL-ACCEPTING CHEMOTAXIS
>TLPC_BACSU_395_416	QTNLLALNAAIEAARAGEQGKG	METHYL-ACCEPTING CHEMOTAXIS

Figure 3.5: Labels of SwissProt proteins containing the motif “ALNAA..AA..A”, and the 23–amino acid region centered around the seqlet “ALNAA..AA..A”; the two numbers following the label of each protein correspond to the beginning and ending offsets of the region that is shown.

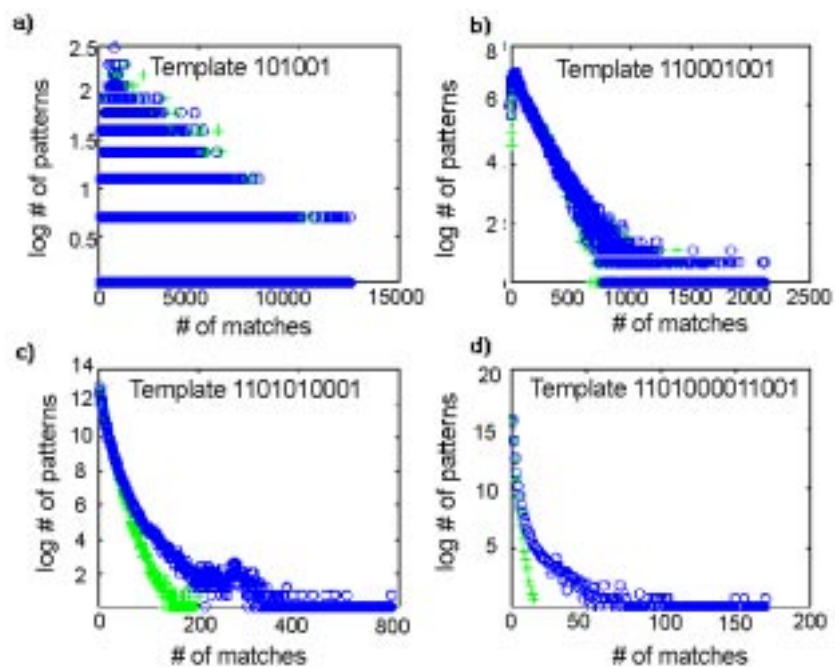


Figure 3.6: Distribution of patterns with given template structures in CleanSP (the cleaned-up version SwissProt Rel. 34) and comparison with the random distribution of the same templates. The character “o” is used for the distributions obtained from CleanSP while “+” marks the random distributions. A point (X, Y) in a curve indicates that there are Y patterns (of the given template structure) matched by X distinct sequences.

Database	Value for d	P_{kept}	% covered in cleaned-up	Additional Seqlets	Total Seqlets	% covered in original
the six genomes	1	3,913	68.2%	1,028	4,941	74.0%
	3	513	35.8%			
	5	171	19.6%			
	7	81	12.7%			
Swiss Prot Release 34	1	13,739	85.6%	6,156	19,895	93.2%
	3	1,585	43.4%			
	5	583	28.3%			
	7	326	21.2%			
NCBI's Non-Redund. Database	1	39,595	93.8%	9,750	49,345	98.3%
	3	5,307	52.7%			
	5	2,074	37.9%			
	7	1,101	29.0%			

Table 3.4: The number of subselected seqlets as a function of the choice for the parameter d (see text), for each of the three databases. Also shown are: the percentage of sequences in each cleaned-up database that are covered by the corresponding P_{kept} , the increase in each P_{kept} by also considering the seqlets obtained from treating the redundant groups, the final number of subselected seqlets and the obtained coverage for each original database using all the subselected seqlets.

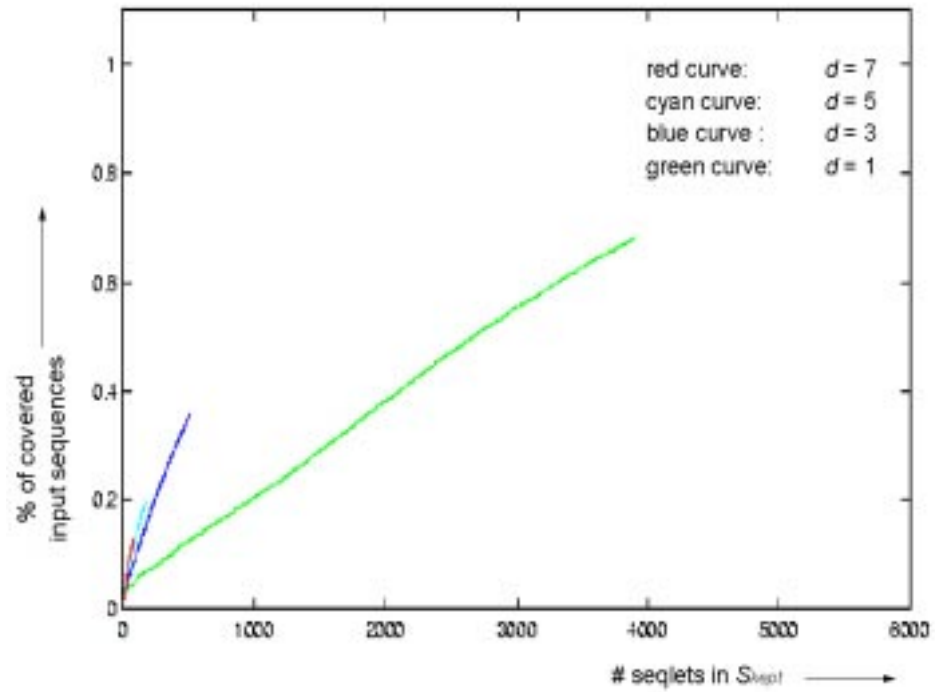


Figure 3.7: Data base of the 6 genomes: coverage (excluding the highly-similar sequences) as a function of the cardinality of P_{kept} .

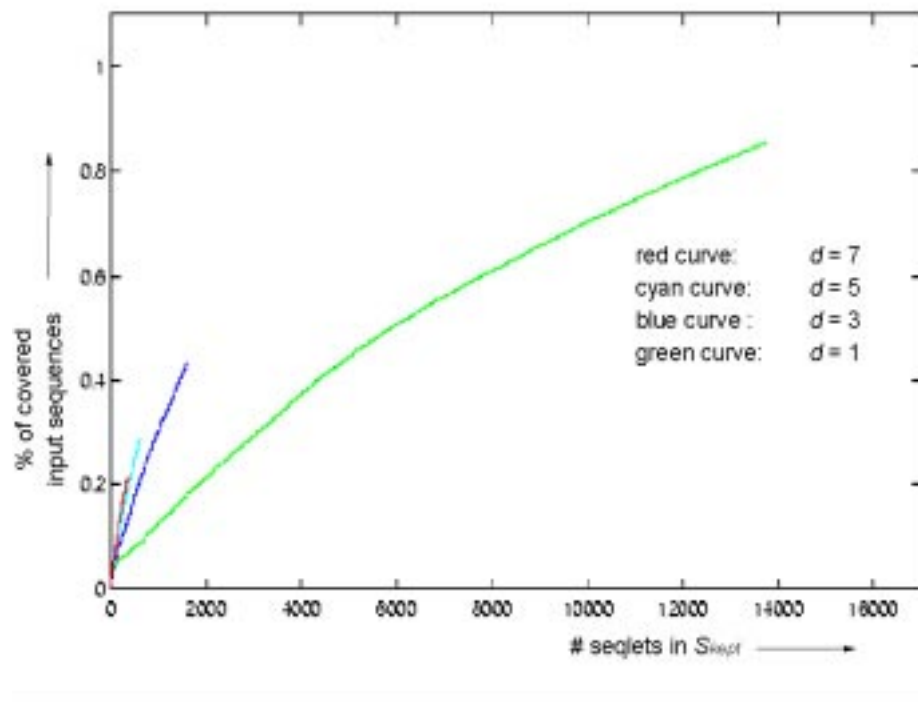


Figure 3.8: SwissProt Rel, 34: coverage (excluding the highly-similar sequences) as a function of the cardinality of P_{kept} .

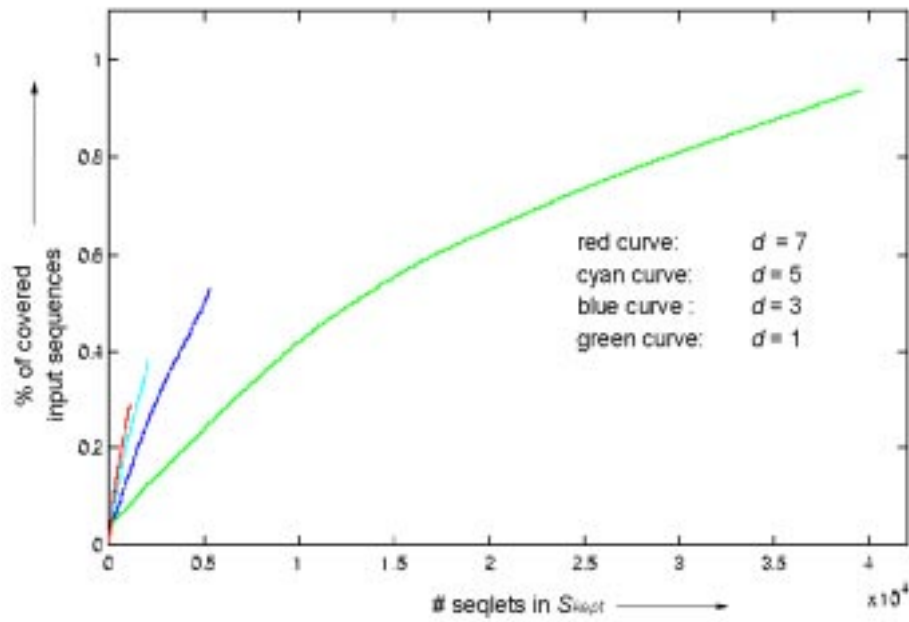


Figure 3.9: NRDB: coverage (excluding the highly-similar sequences) as a function of the cardinality of P_{kept} .

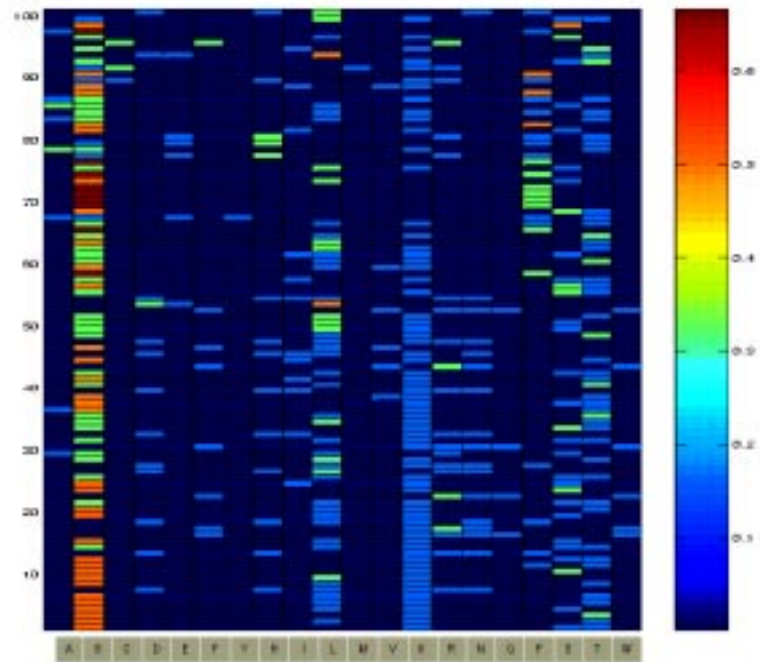


Figure 3.10: NCBI's non-redundant database: the amino acid composition of the first 100 seqlets contained in the P_{kept} that corresponds to $d = 7$ after the highly-similar sequences have been removed

ATP/GTP-binding P-loop variations							
1	G..G.GKST	2	G..G.GKaTL	3	G..G.GKTT	4	G..G.GKS.L
5	G..G.GKT.L	6	L..G..G.GKbT	8	L..G..G.GKT	9	L..G..G.GK..L
10	G..GSGKS	11	GP.G.GKT	12	G..GSGKcT	14	G....GKSTL
15	L..G..G.GKS	19	G..G.GKS..L	20	L..G..G.GKT	21	G.GKST....L
23	G.SG.GKS	24	I.G..G.GKS	25	G.GKSTL	28	G.GK.TL...L
31	L.G....GKST	33	G...SGKST	34	G....GKdTL	35	G.GKTT....L
36	G..G.GKT..A	37	G..GSGKT	38	V...G..G.GKT	40	G....GKTTL
41	I.G..G.GKT	42	G....GKST.L	44	I...G..G.GKT	48	G.GKTTL
49	G.GKS.L...L	50	G....GKS.LL	51	L.G....GKT.L	55	GSGKST
56	G..GSG.ST	57	I.G....GKST	59	V.L.G..G.GK	60	L.G....GKTT
61	I.G....GKS.L	62	G....GKT.LL	63	L.G..G.G.T.L	64	G....GKTT.L
66	GP...GKT.L	68	G.SG.G.ST	73	G..G.G.S.LL	75	L.G....GKS.L
78	G.GKT..A...A	81	I.G.SG.GK	83	G.GKT....LA	84	L.eP.G.GKT
86	GP...GKT..A	88	V.I.G..G.GK	92	TGSGKT	94	I.G....GKTT
96	G...SGKS..L	98	G.SGSG.S	99	G.fKSTL...L		

Table 3.5: The 100 seqlets with the highest support (see also the table on the next page). Wherever possible, the seqlets within a category were aligned with respect to one another. The lower case italics were used for convenience and are place-holders for the following bracketed expressions: **a**: [STGDAR], **b**: [STGDK], **c**: [STGDKY], **d**: [STGK], **e**: [GASMDL], **f**: [GSETV], **g**: [LIVMFY], **h**: [LIVMF], **i**: [LIVMA], **j**: [LIVMC], **k**: [LIVMF], **t**: [ILVMF], **m**: [QKCS], **n**: [KRQA], **o**: [IVTNF], **p**: [QKASN], **q**: [QKIAGN], **r**: [RKAHQN], **s**: [KRQNE], **t**: [KRQMN], **u**: [LFYIMS], and **v**: [AGSPE]. A bracket indicates a position that can be occupied by any one of the residues in the bracket.

Protein Kinase active site	Collagen P/G repeats	Homeobox DNA-binding	ATP-binding, kinases
7 HRDgK..N.L	58 G.PG..G.PG	16 WFQN.R.K	46 LG.G.FG.V
13 HRDhKP.N	65 G..G.PG..G.P	17 WFmNRR.K	85 LSGG.....A.A
18 H.DiKP.N.L	69 G.PG..G..G.P	22 WFQNRR	Nuclear hormone receptors
26 HRDL...N.L	70 G..GP.GP.G	30 WFQN.R.n.K	91 CRu.KC...GM
27 RDjKP.N.L	71 G.PG.PG..G	43 V.oWFpNRR	95 C.vC..FFRR
32 I.HRDlK..N	72 PG..G.PG..G	52 QV.oWFqNR	Various ATP-binding
39 HRDkK..NI	74 G..G.PG.PG	Zinc fingers	53 L.LDE....LD
45 I.H.DlK..N.L	76 GP.G..G.PG	77 H.R.H.GEsP	93 L.LDE.T..L
47 HRDLK..N	82 PG.PG..G.P	79 H...HTGtP	Protein kinases
54 IHRD...N.L	87 G.PG.PG...P	80 H.R.HT.E.P	67 GT..Y.APE
100 H.DL.P.N.L	90 PG.PG...PG	89 R.H.G.KP..C	ABC Transporter
	97 G..G..GAPG		29 LSGG...R...A

Table 3.5: Continuation of the table from the previous page.

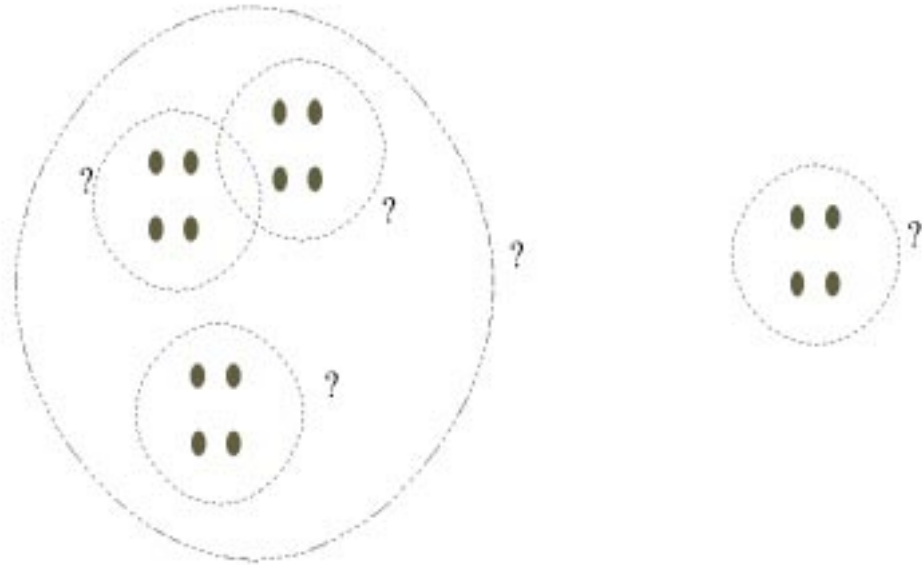


Figure 3.11: The difficulty with clustering: it is not clear whether the 16 items above belong to 2, 3, 4 or 16 natural clusters. Although one could decide on a value for the distance threshold that would produce the correct answer for this particular dataset, the value will not necessarily produce the correct results on all possible datasets.

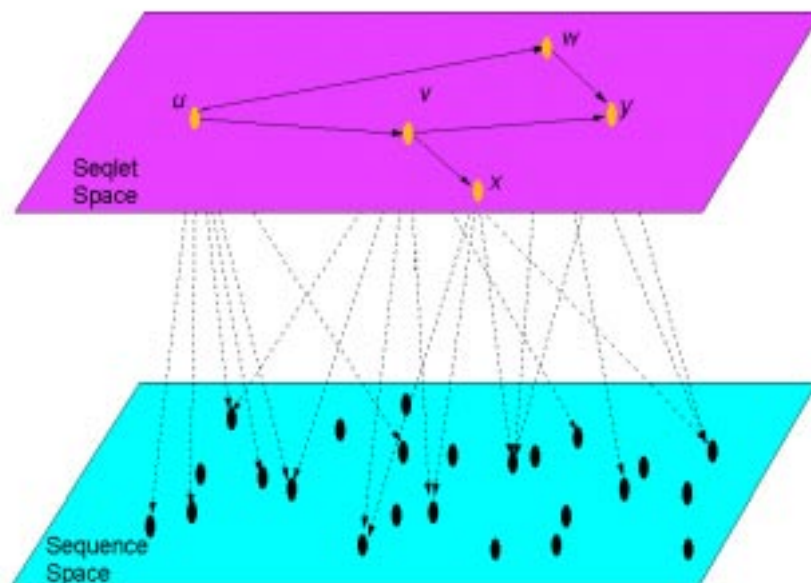


Figure 3.12: A schematic representation of the relations between the seqlets and the sequences in the space of proteins.

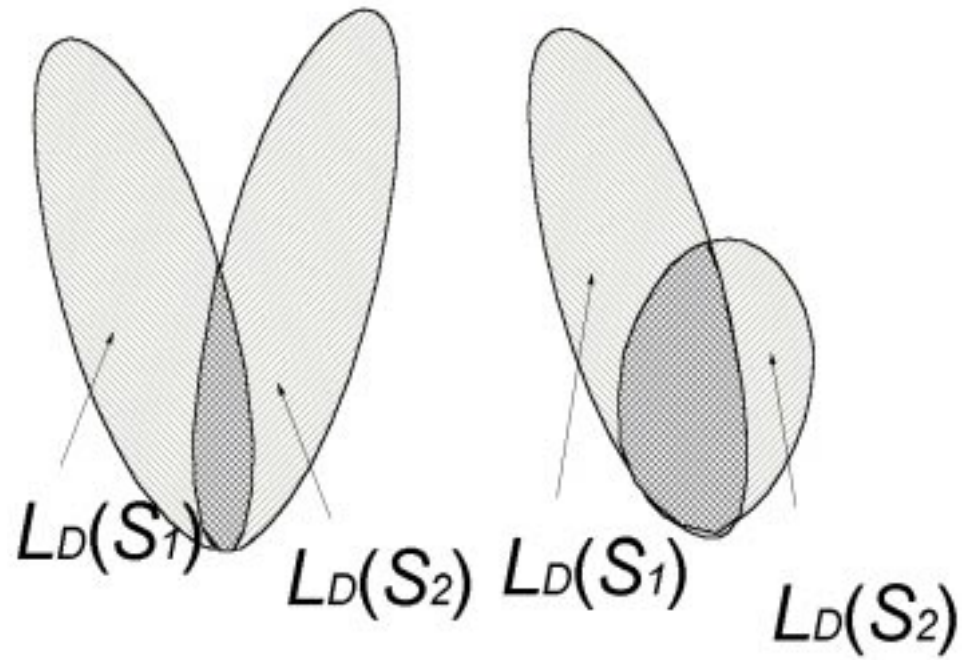


Figure 3.13: Intersecting seqlets and the respective offset lists. Left: the intersection is but a small percentage of either set. Right: the intersection is a substantial fraction of one of the sets.

Chapter 4

Sequence Homology Detection

We next describe a new approach for identifying sequence similarity between a query sequence and a database of proteins. The central idea is to use a the set of seqlets obtained from the underlying data base through an off-line computation. These seqlets are subsequently searched for in every query sequence presented to the system. A seqlet matched by a region of the query pinpoints to a local similarity between that region and all the database sequences also matching that seqlet. In a final step, all such local similarities are examined further by aligning and scoring the corresponding query and database regions. By using a set of prudently chosen seqlets, the tool presented in this work is able to discover weak but biologically significant similarities. We provide a number of examples using both classified and unclassified proteins that corroborate this claim. Furthermore, the performance is superior to other existing methods since it depends on the size of the seqlet set used and not on the underlying database. This last feature is of increased importance given the rate of accumulation of genomic data.

4.1 Introduction

The first step following the sequencing of a new gene is an effort to identify that gene's function. The most popular and straightforward methods to achieve that goal exploit the *first fact of biological sequence analysis* (see Section 1.2): if two peptide stretches

exhibit *sufficient* similarity at the sequence level (i.e., one can be obtained from the other by a small number of insertions, deletions and/or amino acid mutations) then they are likely to be biologically related [69, 28, 20, 83]. Within this framework, the question of getting clues about the function of a new gene becomes one of identifying homologies in strings of amino acids: one is given a query sequence Q (the new gene) and a set D of well characterized proteins and is looking for all regions of Q which are similar to regions of sequences in D .

The first approaches [76, 97] used for carrying out this task were based on the technique of *dynamic programming*. In particular, a query system is compared with all the database sequences through pairwise optimal alignments. Unfortunately, the computational requirements of this method quickly render it impractical, especially when searching large databases (as is the norm today). Simply put, the problem is that dynamic programming variants spend a good part of their time computing homologies which eventually turn out to be unimportant. In an effort to work around this issue, a number of algorithms have been proposed which focus on extensive local similarities only. The most well known among them are *FASTA* [72, 84] and *BLAST* [3, 4]. In the majority of the cases, increased performance is achieved by first looking for *ungapped homologies*, i.e. similarities due exclusively to mutations and not insertions or deletions. The rationale behind this approach is that in any substantial *gapped* homology between two peptide strings chances are that there exists at least a pair of substrings whose match contains no gaps. Locating these substrings (the ungapped homology) can then be used as the first step towards obtaining the entire (gapped) homology.

Identifying the similar regions between the query and the database sequences is only the first part (the computationally most demanding) of the process. The second part involves the evaluation of these similarities, in order to identify which among them are substantial enough to sustain the inferred relation (functional, structural or otherwise) between the query and the corresponding database sequence(s). Such evaluations are usually performed by combining biological information and statistical reasoning. Typically, similarity is quantified as a score computed for every pair of related regions. Computation of this score involves the use of gap costs (for gapped alignments) and of appropriate mutation matrices giving the evolutionary probability of any given amino acid changing into another (e.g. the PAM [25] and BLOSUM [45] families of matrices).

Then, the statistical importance of this cost is evaluated by computing the probability (under some statistical model) that such a score could arise purely by chance [61, 60]. Depending on the statistical model used, this probability can depend on a number of factors as the length of the query sequence, the size of the underlying database etc.

No matter, however, what statistical model one uses there always exist the so called “gray areas”, i.e. situations where a statistically unimportant score indicates a biologically important similarity. Unfortunate as this might be, it is also inescapable; there is after all a limit to how well a statistical model can approximate the biological reality. And for as long as we lack hard biological rules to guide our inferences from the sequence domain to the function/structure domain, statistics will probably keep on playing a central role in deciding which homologies are to be considered important and which not.

Given these restrictions, it seems that the task of bringing more of the gray area similarities under the light is synonymous to the task of devising better (more biologically relevant) statistical models. Here we try to achieve this goal by introducing *memory* into our calculations. Existing statistical frameworks are memoryless; whenever a homology between a region A of the query sequence and a region B of some database sequence is found, the similarity is evaluated without taking into account that A might also be similar to several other database regions. So, although seen in isolation the homology between A and B might seem statistically insignificant, this is certainly not the case when the big picture is considered.

Memory is introduced by using seqlets to identify groups of related oligopeptides that appear unexpectedly many times in the underlying database. Whenever a query sequence is presented to the system, we locate all the query regions matching one or more of these seqlets. Every match acts as a hypothesis of similarity between the query region and all the database regions also matching that seqlet. In a final step all these hypotheses are further examined by aligning and scoring the areas around the corresponding similarity regions. The highest scoring among them are then reported to the user.

The success of the method we propose here depends crucially on our ability to identify a set of seqlets characteristic of the underlying database. Up to now, there were no computational tools powerful enough to handle the task of pattern discovery in data sets of the size of existing protein databases. As a result, analogous efforts [9, 46, 44]

were restricted to patterns characterizing groups of proteins already known to be related. Using TEIRESIAS though, we are able to detect not only family specific patterns but also patterns that cross family boundaries.

4.2 Existing Methods

We will describe here existing methodologies for the task of similarity searching. There exist three well established tools for comparing query sequences with an underlying database: FASTA [72, 84], BLAST [3, 4] and BLOCKS [44, 46]. The first two compare sequences by aligning them. The last, is in spirit similar to the methodology that we propose here: it constructs pattern-like descriptors for protein families and matches them against the given query sequence.

All three of these tools work, essentially, in two distinct phases:

Comparison: The query is pair-wise compared with all the proteins in the underlying database D and a score is assigned to every comparison.

Evaluation: All the comparisons are evaluated according to their score, and the most important among them are reported back to the user.

For each of the three tools, each of these phases is discussed.

4.2.1 FASTA

Comparison

Conceptually, given a query sequence Q and a database sequence s , FASTA computes the entries of the following $|Q| \times |s|$ binary matrix:

$$T[i][j] = \begin{cases} 1, & \text{if } Q[i] = s[j] \\ 0, & \text{if } Q[i] \neq s[j] \end{cases}$$

where $Q[i], s[j]$ are, respectively, the i -th character of Q and the j -th character of s . The diagonals of this matrix register, in essence, all possible ungapped alignments that one can get by *sliding* the sequences Q and s over one another. For example, imagine that

we slide s to the left so that the j -th character of s is aligned with the first character of Q . Then the diagonal of the matrix T that starts at $T[1][j]$ represents this alignment: the entry $T[i][j+i-1]$ is 1 if and only if the i -th character of Q and the i -th character of the string $s[j..|s|]$ are the same. Along the same lines, diagonals “below” the main diagonal of T represent alignments where s is slid to the right of Q (see Figure 4.1).

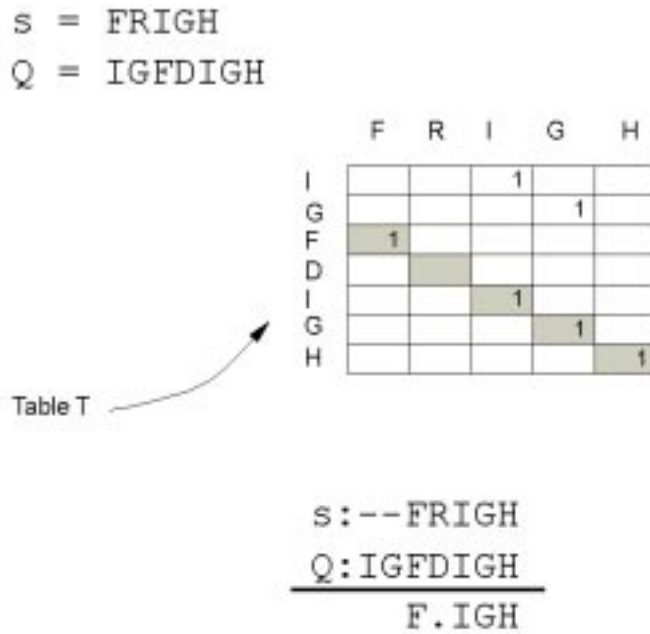


Figure 4.1: Example FASTA table for the sequences s and Q . The diagonal that has been marked indicates the most substantial local, ungapped alignment between s and Q . This alignment is explicitly shown below the table.

Computing T in a brute force manner requires $|Q||s|$ time. FASTA avoids this expensive computation by building a hash table H for the query sequence Q : the entries of H are labeled with strings of length k_{tup} and the entry labeled x contains all the offsets

within Q where the substring x appears. The parameter k_{tup} is user-defined. For now, assume that its value is 1.

When s is compared with Q , every position j of s is visited in turn. Let y_j be the substring of length k_{tup} that starts at offset j within s . If $H[y_j]$ is not empty, then for all $i \in H[y_j]$ the table entry $T[i][j]$ is set to 1.

Substantial local alignments (of the ungapped variety) can be detected as stretches of closely spaced ‘1’s along the diagonals of T . FASTA identifies such stretches and scores them by using a user-defined mutation matrix (usually a member of the PAM or BLOSUM families). Notice that identifying the stretches of ‘1’s seems to require scanning the entire table T , thus negating the benefits of utilizing the hash table H . This is avoided by using a smart implementation of T . In particular an array L_T of lists is used, with one list per diagonal of T . When the offset i of Q is found to be in the currently checked entry $H[y_j]$ of the hash table, then the $(j - i)$ -th entry of L_T is updated by adding the appropriate element at the tail of the resident list (notice that $(j - i)$ may be negative; this is not a problem as the resulting array index can be easily recalibrated by adding the appropriate constant). Thus, locating the substantial local alignments reduces to traversing entries of L_T .

Consider now the running time of the above approach and in particular the time needed to compute the entries of the matrix T (since this is the dominating factor). For every j in the database sequence s the required time is proportional to the contents of $H[y_j]$. If $k_{tup} = 1$, then assuming a uniform distribution of amino acids in the sequence Q one expects to find about $|Q|/20$ offsets in every entry of the hash table H . So, the time needed to treat all input sequences $s \in D$ is:

$$O(|Q| + \frac{|Q| \sum_{s \in D} |s|}{20}) = O(|Q| + \frac{|Q||D|}{20}),$$

where the first additive factor indicates the time needed to build H .

The distribution of amino acids is not, of course, uniform but this assumption is not unrealistic for the purposes of exhibiting the benefits of using the look-up table H . The improvement achieved (over the naive $O(|Q||D|)$ time that would otherwise be needed) can be increased by increasing the value of k_{tup} . Again using the uniform distribution assumption it is not hard to see that, in general, the time will be

$$O(\frac{|Q||D|}{20^{k_{tup}}}).$$

There is a catch, though: for any value of k_{tup} , a local alignment will be recognized iff it has at least k_{tup} *consecutive* amino acids preserved in both the query and the database sequence s under examination. So, large values of k_{tup} , while improving the running time, run the danger of missing weak homologies.

Evaluation

Among the stretches of '1's thus examined, the highest scoring is recognized and its score is associated with the pair (Q, s) . Let this score be denoted by C_s (we only index it by the sequence s because Q is fixed for all the comparisons). FASTA will report as homologues of Q the databases sequences s with the highest scores C_s . This is not enough, though. To understand why, notice that even if Q has no homologues in D , scores C_s will still be created for all $s \in D$. So, one needs a way to tell when the highest scores correspond to real homologues or not. To do so, FASTA uses the distribution of all the scores C_s (see Figure 4.2). Strong homologies will appear as clear deviations from the tail of the distribution.

For weaker homologies, where the above criterion does not give a clear answer, the following approach is used: let C_s be a score of questionable importance. The sequence s is shuffled a number of times (the default value is 50) and each shuffled version s' of s is compared against Q in the way used for the real database sequences. Then C_s is compared with the distribution of the scores $C_{s'}$. If C_s is more than a given number of standard deviations above the mean of $C_{s'}$, then C_s is deemed important.

In a final step, the top scoring alignments are further refined by re-aligning them and re-scoring them, now allowing gaps.

As a closing remark, we note that the score evaluation method described above was used in the original version of FASTA. Later on, when analytical descriptions for the distribution of the scores C_s became available [60, 61, 27], FASTA implementations begun using them. Since these analytical descriptions were pioneered by BLAST, we discuss them in the next section.

4.2.2 BLAST

BLAST (and its recent variants gapped-BLAST, ϕ -BLAST and ψ -BLAST) is the most widely used tool for performing similarity searches. The reason is two-fold:

- it provides a rigorous statistical model [60, 61, 27] for evaluating the significance of local, ungapped alignments with a given score, and
- it can compute these alignments very fast.

In explaining the operation of BLAST, it helps to talk about the evaluation process first.

Evaluation

In the terminology of BLAST, any pair of amino acid strings with the same length is called a *segment pair*. Given any such pair of strings $X[1..m], Y[1..m]$ and a substitution matrix $M[i][j]$ providing the mutation costs between the i -th and the j -th amino acid (where the amino acid ordering is arbitrary), the score of transforming X to Y is defined in a straightforward way as

$$C_M(X, Y) = \sum_{i=1}^m M[X[i]][Y[i]].$$

Consider any pair of sequences Q and s and any positive integer $L \leq \min\{|Q|, |s|\}$. Then there exist

$$(|Q| - L + 1)(|s| - L + 1)$$

segment pairs that can be formed by choosing pairs of length- L substrings, one from Q and one from s . Let now Q_i^L, s_i^L denote the length- L substrings that start at offset i in Q and s respectively. Define the *maximal score* $C_{max}(Q, s)$ of the pair Q and s as

$$C_{max}(Q, s) = \max_{L, i, j} \{C_M(Q_i^L, s_j^L)\}$$

The *maximal segment pair* (MSP) for the sequences Q and s is defined as any pair of equal length substrings Q_i^L, s_j^L for which $C_M(Q_i^L, s_j^L) = C_{max}(Q, s)$.

What Karlin and colleagues in [60, 61, 27] have found is a limiting distribution for the MSP scores: these scores obey an *extreme value* distribution. More specifically, given (i) the probability of occurrence of every amino acid, and (ii) a substitution matrix M , the work mentioned above describes how to compute constants λ and K such that when two random sequences Q ($|Q| = m$) and s ($|s| = n$) are compared:

- the probability that $C_{max}(Q, s)$ is equal or larger than a score S is no larger than

$$1 - e^{-r} \tag{4.1}$$

where $r = K m n e^{-\lambda S}$.

- the probability of finding c or more segment pairs with score at least S is no larger than

$$1 - e^{-r} \sum_{i=0}^{c-1} \frac{y^i}{i!}$$

Given now a query sequence Q and a database sequence s , the expression above allows the evaluation of the statistical significance of $C_{max}(Q, s)$ (in fact, BLAST assumes that s is the concatenation of all the sequences in the underlying database D ; this allows to take the overall size of D into account, an important parameter that was missed by previous analyses). Locating the MSP which achieves the score $C_{max}(Q, s)$, though, is an expensive task that would require time $O(|Q||s|)$. So, BLAST resorts to an approximation approach: it will find and score all segment pairs that have a particular property (to be discussed next). The hope is that, with high probability, a MSP will be among these segment pairs.

Computation

BLAST makes use of two parameters: w (a string length) and T (a score). They will be explained as they are encountered. For now, it is enough to mention that both the sensitivity of BLAST (i.e. the probability of actually finding a MSP) and its running time are dependent on the values of w and T . In the following discussion we assume that a substitution matrix M has already been chosen and is used for the computation of all scores.

Given a query sequence Q , BLAST begins by first computing the *word list* $WL(Q)$ of Q . This list contains *all* the length w residue strings s_w such that:

$$C_M(Q_i^w, s_w) \geq T, \quad \text{for some } i, 0 \leq i \leq |Q| - w + 1$$

Computation of $WL(Q)$ involves scanning Q and at every offset i compute the *neighbors* of Q_i^w , i.e. all the length w strings whose similarity with Q_i^w is at least T .

The next step is to scan the database D for all the instances of the strings in the word list of Q . Every such instance is called a *hit* and it denotes an area of potential similarity between Q and the database sequence where the hit occurs. In order to be able to scan the database quickly, an automaton is built allowing the concurrent match of all the strings in $WL(Q)$ [54]. When a hit is found in a sequence $s \in D$, the local alignment between Q and s implied by the hit is extended both left and right, in an effort to find a locally maximal segment pair. To speed up the whole process, extension in one direction is terminated when the score falls a certain amount below the best score computed for shorter extensions. Finally, the highest scoring local alignments found in this way are reported back to the user, each one accompanied by its corresponding *p-value* i.e. the probability computed from equation (4.1) for the score S of the alignment.

BLAST only computes segment pairs that (i) contain two words of length w with a similarity score of at least T , and (ii) can be captured by the extension process mentioned above. The question, then, becomes: what is the probability that no MSP is among such segment pairs? An analytical answer to this question does not exist. From simulations described in [3], it appears that this probability directly depends on w and T . In particular, it becomes smaller for large w and small T . This is expected, as such settings allow the consideration of more segment pairs.

Naturally, large values of w and/or small values of T make the run time suffer. The reason for this is that the size of the word list (which dominates the running time) depends exponentially on w, T . In an example taken from [3], a 30-residue sequence generates word lists of size 296, 3561 and 40939 for $w = 3, 4$ and 5 respectively. So, deciding the particular values for w and T is a compromise between execution time and sensitivity. Heuristics for setting these parameters are described in [3].

4.2.3 BLOCKS

BLOCKS [44] generates and uses profile-like descriptors for groups of (known to be) related proteins (the family classification used by BLOCKS is the one described in the PROSITE database). The generated descriptors are called *blocks* and can be searched on any query sequence Q . If a block B from a family of proteins F is found in Q (the semantics of “found” will be described shortly) then this is an indication (depending on

how strong the match is) that Q is a member of F .

Computation

BLOCKS starts with an one-time off-line computation: given a set

$$\mathcal{F} = \{F_1, F_2, \dots, F_n\}$$

of protein families F_i , it computes for every F_i a set $\mathcal{B}_i = \{B_1^i, B_2^i, \dots, B_{j_i}^i\}$ of profile-like blocks B_r^i . The set \mathcal{B}_i is such that there exist at least n_i sequences in F_i containing all the blocks of \mathcal{B}_i and in these sequences the blocks (i) appear always in the same order, and (ii) do not overlap. The number n_i is determined experimentally but is always at least $|F_i|/2$.

A block is as a local alignment generated around a pattern with exactly 3 characters. This pattern is called the *generator* of the block. Figure 4.3 shows such a block and its corresponding generator. Every block gets assigned a score according to how well every individual column is conserved using the methodology of [95].

The block-computation process for each F_i starts by computing all generator patterns with 3 amino acids and support at least n_i . For that purpose, the pattern discovery algorithm of [95] is used. Among these patterns, the 50 highest scoring are kept for further processing. This processing constitutes of (i) merging together generator patterns that overlap consistently, (ii) extending the local alignments induced by the resulting patterns left and right, and (iii) for every pattern, keeping the highest scoring among all the possible extensions.

The above process results in a number of blocks which will be usually overlapping in complex ways. In order to subselect those that will form the block set \mathcal{B}_i , a directed acyclic graph is built. The vertices of the graph are the blocks computed above. An arc is drawn from a vertex B to a vertex B' if there are at least n_i sequences where B appears before B' and with no overlaps. Edges are weighted using a variety of criteria including the scores of B and B' , the total number of sequences containing B before B' etc. Finally, a *best path* is computed and the blocks in this path form the set \mathcal{B}_i . Along with each block the smallest and largest distance from the previous block is saved. These distances are computed from the sequences of F_i that contain all the blocks of \mathcal{B}_i .

Computation

The blocks B_r^i of \mathcal{B}_i are evaluated using the SwissProt version upon which the PROSITE database used was keyed. Every block is transformed into a profile, using the composition of amino acids in every column and this profile is “slid” over every sequence s in SwissProt. For all such s the highest score is recorded (see also Figure 1.18). The scores thus computed are then divided into two groups: those corresponding to the members of F_i (the true positives) and those corresponding to the rest of the sequences (the true negatives). The 99.5% percentile of the distribution of the latter scores becomes the *lower mark* of B_r^i while the median of the former score distribution is called the *upper mark*. These scores are used when evaluating the match of B_r^i with a query sequence Q . If the score resulting from sliding the weight–matrix of B_r^i over Q is above the upper mark, then the match is considered important. If it is below the lower mark it is not. For scores in between the lower and upper marks more information is needed, e.g. if Q fits well the profiles of the other blocks in \mathcal{B}_i , if the order in which the blocks appear in Q is the same as the one prescribed by \mathcal{B}_i etc.

4.3 Motivation and Definitions

The homology search tool that we propose here uses descriptors (in the spirit of BLOCKS) capable of representing related groups of proteins. In our case these descriptors are patterns. Unlike BLOCKS, however, our descriptors are obtained from mining the underlying database in an unsupervised manner. The motivation behind using patterns for describing related polypeptides has already been discussed. In particular, it is known that there is a number of basic elements (either of structural nature like α -helices, β -strands, loops etc. or larger, functional units like domains which are the building blocks that proteins are made of. As explained in Section 1.2, one of the key mechanisms used by evolution to differentiate among species is the mutation of amino acid positions in a protein sequence. Functionally and structurally important regions, though, are more resistant to such mutations. It is then reasonable to expect that such biologically related polypeptides can be identified by discovering conserved positions in their primary structure and an increased degree of reusability. In our terminology, these properties

correspond to patterns with unexpectedly high support.

The approach we propose here is based on the assumption that patterns that appear unexpectedly often in a large database bear some biological significance. These patterns are then essentially used as indices in the underlying database, pinpointing to similarities between a query sequence and the sequences residing in that database. More specifically, the proposed methodology is composed of two distinct phases: *information gathering* and *searching*.

Information gathering: First, and before any search is performed, the underlying database D is mined. During this step, all the *statistically* significant $\langle L, W \rangle$ seqlets are gathered and each such seqlet P is associated with its offset list $L_D(P)$. The notion of “statistical significance” for our purposes, has already been explained in Section 3.3.3.

Searching: The second step is the actual search. Given a query sequence Q , we identify all the patterns P (among those collected in the first phase of the process) which are matched by Q . For every such P , we pair together the region(s) of Q which match P with the corresponding regions of all the database sequences that also match P (these regions are easily accessible through the offset list $L_D(P)$). Finally, the paired regions are extended and aligned in both directions and scored by the use of a (user-defined) mutation matrix and the highest scoring matches are reported along with the implied alignments.

It is worth pointing out here that the information gathering phase is an one-time, off-line computation over D . The results obtained are stored in a file and used every time that a search session is performed over the database D .

4.4 Implementation

As already mentioned, the methodology we propose consists of two phases: information gathering and searching. Since the information gathering phase was the subject of the previous chapter, here we will focus only on the search phase.

4.4.1 Searching

During this phase, query proteins Q are presented to the system and database sequences $s \in D$ similar to Q are identified and reported back to the user. The searching phase utilizes a set Π of seqlets obtained by mining the input database D . For the purposes of the discussion here it is sufficient to assume that Π is a set of $\langle L, W \rangle$ patterns. Each seqlet $P \in \Pi$ is accompanied by its offset list $L_D(P)$ and has support at least K_{min} in D . The way that the parameters L, W and K_{min} are computed (given the underlying database D) was described in Section 3.3.

Pattern Matching

When a query sequence Q is provided to the system, we first locate all $P \in \Pi$ that are matched by Q . This can be done very fast by using a hashing variation of a technique described in [41]. More specifically, for every position within Q we generate W hash values, one for every substring of length $2, 3, \dots, (W + 1)$ starting at that position. For every such substring the corresponding hash value depends only on the first and last characters of the substring as well as on the number of residues in between these two characters. Table 4.1 below provides an example of the process for a given query sequence.

The hash entry corresponding to a particular value h contains all the offsets p of the query sequence Q such that a substring (of length at most $W + 1$) starting at p hashes to the value h . Figure 4.4 gives an example of the hash table generated for a particular query sequence.

In order to check if a seqlet $P \in \Pi$ is matched by Q we use an array of counters $C[1..|Q|]$ of size equal to the length of Q . Initially, every entry of the array is set to 0. Starting at offset 1 in P , we locate all offsets j within P that correspond to a residue, excluding the offset of the last residue. For every such j , let R be the shortest substring of P starting at j and containing exactly two residues. Let OL denote the list of offsets in Q pointed to by the hash table entry corresponding to R . If OL is not empty, then for every offset $p \in OL$ the counter $C[p - j + 1]$ is incremented by one. If the seqlet P contains exactly m residues, then at the end of this process the counter $C[i]$ will have

$Q = AFGHIKLPNMKAMGH$		$W = 4, \text{ Position} = 6$
<i>Substring starting at position 6</i>		<i>Hash value</i>
KL	(first_char = K, last_char = L, gap = 0)	1,184
KLP	(first_char = K, last_char = P, gap = 1)	1,601
KLPN	(first_char = K, last_char = N, gap = 2)	1,394
KLPNM	(first_char = K, last_char = M, gap = 3)	1,291

Table 4.1: The hash values generated for the $W = 4$ substrings starting at position 6 of the sequence Q . The hash value used for a substring s is

$$H(s) = ((V(\text{first_char}) - V('A')) + (V(\text{last_char}) - V('A')) * 26) * W + \text{gap}$$

where $V(c)$ is the ASCII value of the character c , first_char and last_char are the first and last characters of s respectively and gap is the number of residues in between the first and last characters of s . Notice that because of the $\langle L, W \rangle$ density restriction gap is always less than W .

the value $(m - 1)$ if and only if Q matches P at offset i ¹. In fact, in order to handle long and dense seqlets with many amino acids and few don't-care characters (such seqlets are typically the result of treating the redundant groups), we allow some flexibility in the process described above. More specifically a seqlet P with m residues, is assumed to match the query Q at offset i when

$$C[i] \geq \min\{L, \lceil 0.8m \rceil\}.$$

¹An advantage of the matching technique described above is that it typically requires time which is sublinear to the size of the query sequence Q and only depends on the number of residues in the seqlet P .

That is, the seqlet is allowed to match *approximately* at a given offset. Approximate matching counterbalances the potential *overfitting* of the data, a problem known to be associated with long patterns (see Section 2.6).

Chaining and Scoring

Once a seqlet $P \in \Pi$ is found to be matched by a substring of Q starting at offset i , we relate that substring of Q with all other database regions also matching P . This is easily done by scanning the offset list $L_D(P)$ which contains exactly these regions. More specifically, each entry $(j, k) \in L_D(P)$ indicates that the substring starting at offset k of the j -th database sequence s_j is an element of $\mathcal{L}(P)$. The local similarity between the query sequence Q and the database sequence s_j is then registered as a quadruplet (i, j, k, l) , called a *segment*, which gets associated with s_j . The number $l = |P|$ is the length of the local similarity.

Sometimes, two distinct seqlets P and P' matching both Q and a database sequence s_j correspond to the same local similarity between Q and s_j . An example of such a situation is depicted in Figure 4.5. In such cases, the individual segments corresponding to the two seqlets must be chained into one. In particular, two segments (i, j, k, l) and (i', j, k', l') associated with s_j are called *compatible* if and only if:

$$k \leq k' \quad \text{and} \quad k + l + w_Len > k' \quad \text{and} \quad k' - k = i' - i$$

where w_Len is an integer parameter defined by the user; w_Len allows for the chaining of segments which are not intersecting, as long as one starts no more than w_Len positions after the end of the other. The segment resulting from chaining (i, j, k, l) and (i', j, k', l') together is

$$(i, j, k, \max(l, k' - k + l'))$$

Chaining of compatible segments takes place every time that a new segment is associated with a database sequence s_j , as the result of locating a seqlet $P \in \Pi$ matched by both Q and s_j . If there exist segments already associated with s_j which are compatible with the newly arriving segment then the relevant pair of the new and the existing segment is discarded and replaced by the outcome of their chaining.

Having identified all the local similarities between Q and the database sequences we are left with the task of evaluating these similarities. This is done by assigning a score

(using a user-defined scoring matrix) to every database sequence s_j that is associated with at least one segment. Several options are available for the scoring function. One approach is to score each segment of s_j individually and assign to s_j the highest of these scores. Scoring a segment (i, j, k, l) can be done in either of two ways

- no gaps allowed: in this case the score is computed from the ungapped alignment implied by the segment, namely the alignment of the regions $Q[i, i + l - 1]$ of the query and $s_j[k, k + l - 1]$ of the sequence. Furthermore, the user is given the option to extend the alignment “around” the segment by setting the variable *extend*. If this value is greater than 0 then the score is computed from the ungapped alignment of the regions $Q[i - extend, i + l - 1 + extend]$ and $s_j[k - extend, k + l - 1 + extend]$.
- allowing gaps: this option is available only when $extend > 0$ and permits for a finer scoring of the area around the segment, by allowing for gaps in that area of the alignment.

Other scoring options are also offered, taking into account the relative order of the segments associated with the database sequence s_j currently being scored. After scoring each segment individually as described above, one possibility is to build a directed, weighted graph. The vertices of this graph are the segments associated with s_j and there is a directed line between the segments (i, j, k, l) and (i', j, k', l') if

$$i \leq i' \quad \text{and} \quad k \leq k'.$$

Every vertex is assigned a weight equal to the score of the corresponding segment while every edge is weighted based upon (i) how close the two segments are, i.e. the value of $(i' - i - l)$, and (ii) how regular is the displacement among the two segments, i.e. how much different $(i' - i)$ is from $(k' - k)$. The score of a path within this graph is the sum of the weights of all the vertices and edges of the path. The path with the maximal score is then computed and that score is assigned to S_j .

4.5 Results

In this section we discuss the results of the proposed methodology when applied to a test database, in this case SwissProt Rel. 34. We begin by recapitulating the outcome

of the information gathering phase on SwissProt and continue with the presentation of a few example homology searches. In particular, we will consider two query sequences one of which was up to now of unknown functionality.

4.5.1 Information Gathering

The treatment of SwissProt involves cleaning up the database, identifying the redundant groups, deciding on the values of the parameters L , W and K_{min} , running TEIRESIAS on the clean database, and finally, augmenting the seqlets found in the clean database with the seqlets obtained from the pattern discovery process on the redundant groups (for details see Sections 3.3 and 3.4). Table 4.2 below summarizes the results of the clean-up process performed on SwissProt.

Sequences /aa in original DB	Sequences in redundant groups / Redundant groups	Sequences / aa in cleaned-up DB
59,021 / 21,210,388	40,407 / 9,165	27,779 / 10,596,414

Table 4.2: Results of the clean-up process in SwisProt, Rel. 34.

For the parameter settings $L = 6$, $W = 12$, $K_{min} = 15$, the final set we obtain contains 565,432 seqlets. The coverage achieved on SwissProt by these seqlets is described in Table 4.3. Notice that for long seqlets we allow for approximate matching, along the lines described in Section 4.4.1. Figure 4.6 gives distributions for the following characteristics of the patterns in Π (i) length and (ii) number of amino acids.

As shown in Table 4.3, one of the key goals for the success of the search phase to follow (namely the good coverage of SwissProt) has been achieved. A second important question is whether the patterns discovered have biological significance. To address this concern we analyzed the most frequently occurring among these patterns. The resulting annotation is almost identical to the one presented in Table 3.5 of Section 3.4. The only

Total number of seqlets	Number of proteins covered	Number of amino acids covered
565,432	57,983	12,567,345

Table 4.3: Coverage of the entire SwissProt database by the patterns generated in the information gathering phase. The amino acids *covered* by a pattern are exactly those that belong to substrings matching the pattern. Notice that for dense and long patterns (coming mostly from the processing of the redundant groups) we have allowed for approximate matches, where “most” of the pattern (specifically, 80% of the patterns’s residues) is matched by a region. It is worth pointing out that most of the uncovered sequences are fragments. More specifically, only 231 contain more than 50 amino acids.

difference is the relative order of the seqlets and the exact form of their variability. The functional groups identified, though, are the same.

From this analysis it is evident (at least for the examined seqlets) that the pattern discovery process identifies sequence features that are biologically important. In all fairness it should be mentioned that not all the discovered seqlets exhibit such clear cut functional specificity. Several of them correspond to regions (e.g. loops, coiled-coils, transmembrane) which are traditionally considered uninteresting at least from the standpoint of functional annotation. Occasionally, though, even such weak similarities can provide useful hints for characterizing protein regions. We have implemented two mechanisms that allow the exploitation of this potential. First, the user is provided with the list of all the patterns which are matched by the query sequence. An expert user will, in most cases, be able to identify which patterns are of biological importance. Selection of a particular pattern leads to a scoring refinement, focusing only on the areas of the database covered by this pattern. Second, when the underlying database includes annotations of various sequence regions, this annotation is used in conjunction with the patterns for the extraction of useful information. Examples of the use of these two

mechanisms are given in the next subsection.

4.5.2 Searching

In order to showcase the searching phase (and to explain how it should be used) we selected two query sequences. The first is a well studied and annotated core histone 3 protein (SwissProt ID: *H31_HUMAN*) while the second is a not yet characterized open reading frame (SwissProt ID: *YZ28_METJA*) from *Methanococcus Jannaschii*.

H31_HUMAN

The release 34 of the SwissProt database contains 33 sequences which are annotated as Histones 3, among which and *H31_HUMAN*, the core histone 3 protein found in humans. The top-scoring results of searching this sequence with our homology detection tool are tabulated in Table 4.4. The scores mentioned in that table are obtained using the PAM 130 matrix [25] and every matching sequence from the database is assigned the score of its highest scoring segment.

All 33 core histones 3 of SwissProt Rel. 34 are correctly identified as homologous to *H31_HUMAN*. Furthermore, several other proteins (*YB21_CAEEL*, *CENA_HUMAN*, *CSE4_YEAST*, *YL82_CAEEL*, *CENA_BOVIN*, *YMH3_CAEEL*) are found to have extensive local similarities with *H31_HUMAN*. Inspection of the annotation for these proteins indicates that they are known histone 3-like proteins. As a final note, *H3_NARPS* (a known histone 3) appears within the release 34 of SwissProt only as a fragment and that is the reason that is scored lowest in the list of results. Figure 4.7 gives a selected view (both high- and low-scoring) of the alignments generated for the query sequence *H31_HUMAN*.

YZ28_METJA

H31_HUMAN is in a sense an easy test case because the database contains several sequences which are highly homologous to it. An interesting question to ask is how our methodology fares when presented with “borderline” sequences, i.e. sequences for which no known homology exists. In an effort to address this question the system was presented with the yet not annotated sequence *YZ28_METJA*, an open reading frame with 1272 residues from the genome of *M.Jannaschii*.

H32_XENLA (298)	H3_ACRFO (297)	H3_CAEEL (291)	H3_VOLCA (289)
H32_MEDSA (288)	H3_ENCAL (288)	H3_CHLRE (287)	H31_SCHPO (286)
H3_PEA (284)	H3_MAIZE (284)	H33_CAEEL (284)	H33_HUMAN (284)
H33_SCHPO (277)	H31_TETPY (274)	H34_CAIMO (272)	H3_EMENI (271)
H3_NEUCR (271)	H3_YEAST (269)	H32_ORYSA (269)	YB21_CAEEL (232)
H3_HORVU (221)	H34_MOUSE (204)	H3_ENTHI (179)	H32_TETAM (177)
H32_TETPY (176)	H33_TETHH (168)	H32_TETBO (153)	H3_LEIIN (110)
CENA_HUMAN(100)	CSE4_YEAST (96)	YL82_CAEEL (86)	CENA_BOVIN (84)
YMH3_CAEEL (79)	H3_NARPS (64)		

Table 4.4: High scoring homologies between *H31_HUMAN* and the SwissProt sequences. Next to each sequence we give the similarity score (using the scoring table PAM 130) of the highest scoring local alignment between that sequence and *H31_HUMAN*.

The top scoring alignments produced by our system when presented with this query sequence are shown in Figure 4.8.

For the purposes of functional annotation of *YZ28_HUMAN*, the above results are not very enlightening as the database hits involve quite diverse proteins: the first two (*NTNO_HUMAN*, *NTNO_BOVIN*) are sodium-dependent noradrenaline transporters while the last one (*KAPL_APLCA*) is a kinase.

With these questions in mind, we proceeded to a closer examination of the similarities between *YZ28_METJA* and the database sequences. For this analysis every pattern matching *YZ28_METJA* was scrutinized individually. As mentioned at the end of Section 4.5.1, the search phase allows the user to select any of the patterns matched by the query sequence at hand and focus on the local alignments induced by that particular pattern alone, disregarding all the other patterns. This feature was employed for each of the patterns matched by *YZ28_METJA*. The intention was to discover if any such pattern is specific to a particular protein family, thus giving clues about the functionality of *YZ28_METJA*.

As it turned out, there exist three patterns (namely the patterns “Y..S..I...DLK”, “NIL.....IKL” and “I.H.DLK.....D”) which are very specific for the kinase family. Figure 4.9 describes a few among the top scoring alignments produced for the first one of them while Table 4.5 contains a complete listing of all the database sequences containing that particular pattern. Tables 4.6 and 4.7 give the corresponding listings for the remaining two patterns. Figure 4.10 graphically represents the distribution of all the patterns matched by *YZ28_METJA* and the areas covered by the three kinase-specific patterns.

The pattern “Y..S..I...DLK” generates 24 hits within SwissProt. All of these proteins (with the exception of *NABA_RAT*, a sodium/bile acid cotransporter) are annotated as protein kinases (two of them, *KD82_SCHPO* and *KKK1_YEAST*, are characterized as putative/probable kinases) with the majority belonging or showing similarity to the serine/threonine kinase family. Furthermore, “Y..S..I...DLK” not only belongs to the kinase domain of these proteins but it actually contains the active site (aspartic acid *D*) of that domain.

Similar results (Table 6) are obtained for “NIL.....IKL”, the second of the three patterns. In this case the number of database hits is 34 and all of them (excluding two unannotated ORFs from Yeast and *Mycoplasma Hominis*) are known (or probable) protein kinases.

Finally, the third pattern “I.H.DLK.....D” generates 30 SwissProt Rel. 34 hits, all of them known or putative protein kinases. Furthermore, as in the case of the first of the three patterns, the pattern “I.H.DLK.....D” includes the active site of the kinase domain.

It is interesting to notice that all three of the aforementioned patterns are specific instances of (parts of) the following general pattern:

$$[LIVMFYC].[HY].D[LIVMFY]K..N[LIVMFYCT][LIVMFYCT][LIVMFYCT].$$

MP38_MOUSE	MKK2_DROME	MP38_XENLA	KRAF_CAEEL	DAPK_HUMAN
PKX1_HUMAN	KAPC_YEAST	KAPA_YEAST	ASK2_ARATH	KCC1_YEAST
CC28_YEAST	KD82_SCHPO	SPK1_YEAST	SGK_RAT	GCN2_YEAST
FUSE_DROME	NABA_RAT	KAPC_ASCSU	KKK1_YEAST	KGPA_BOVIN
KGPB_HUMAN	KGP3_DROME	KGP2_DROME	KDC2_DROME	

Table 4.5: SwissProt Rel. 34 sequences containing the pattern “Y..S..I...DLK”. All of them are annotated as protein kinases or probable/putative protein kinases (almost exclusively of the serine/threonine variety). The only exception is the protein *NABA_RAT* which is annotated as a sodium/bile acid cotransporter.

This more general pattern is the PROSITE database entry with accession number PS00108, namely the signature of the serine/threonine protein kinase active site. Notice that this PROSITE signature is too specific for picking up a kinase catalytic site in the areas of *YZ28_METJA* covered by the three patterns examined above. This situation is a manifestation of the *overfitting* problem discussed in Section 2.6: there is always the danger that the set of positive examples used (in this case, the specific set of known serine/threonine kinases used by PROSITE) is biased and as a result the features learned (here the kinase signature) while explaining the observations are not general enough to extrapolate efficiently to new instances of the family under consideration (i.e. there are false negatives). The cure for this problem is the use of as large a training set as possible and this is the crux of the approach we propose here. Of course, it is also possible that our methodology creates the reverse problem, that of *underfitting* in which case the patterns discovered explain too liberally the observed data and introduce false positives. Until, though, a definite answer is given by the performance of the appropriate lab tests, we offer the statistical significance and the functional specificity of the patterns examined above as indications that they do correctly model kinase activity.

Using Existing Annotation

Of the 410 patterns matched by *YZ28_METJA*, only the three patterns analyzed above exhibit such clear cut functional specificity. This does not mean that the remaining 407 are useless. As discussed in Section 4.5.1, the kind of biological inference that can be drawn from a local similarity between two sequences is not always of a functional nature. Sometimes the homology indicates preservation of structure and yet other times it might correspond to functional units with a supporting role (e.g. DNA-binding domains) in the overall function of the sequences compared. In an effort to explore such weaker similarities we have provided for a way to exploit the annotation available in the underlying

CC7_SCHPO	CDK2_ENTHI	CDK6_HUMAN	IPL1_YEAST	JKK1_HUMAN
JKK1_MOUSE	KG1Z_YEAST	KKIA_HUMAN	KNQ1_YEAST	KPBG_MOUSE
KPBG_RABIT	KPBG_RAT	KS61_MOUSE	KS62_HUMAN	KS62_MOUSE
KS6A_CHICK	KS6A_XENLA	KS6B_XENLA	MKK2_YEAST	MPK1_HUMAN
MPK1_MOUSE	MPK1_RABIT	MPK1_RAT	MPK1_XENLA	MPK2_HUMAN
MPK2_RAT	MPK2_XENLA	PAK1_SCHPO	PK3_DICDI	PKD1_DICDI
PKX1_HUMAN	ST20_YEAST	YFH8_YEAST	YLI1_MYCHO	

Table 4.6: SwissProt Rel. 34 sequences containing the pattern “NIL.....IKL”. All of them (except from the non-annotated *YFH8_YEAST* and *YLI1_MYCHO*) are protein kinases (known or probable). Again, serine/threonine kinases are the majority.

ASK1_ARATH	ASK2_ARATH	CC2C_DROME	CC2_DICDI	CC2_SCHPO
CDK7_CARAU	CDK7_HUMAN	CDK7_MOUSE	CDK7_RAT	CDK7_XENLA
CTR1_ARATH	FUSE_DROME	GCN2_YEAST	KCC4_YEAST	KD82_SCHPO
KEMK_MOUSE	KFD3_YEAST	KKK1_YEAST	KP78_HUMAN	KPBG_MOUSE
KPBG_RABIT	KPBG_RAT	KPBH_HUMAN	KPBH_RAT	KPK1_ARATH
KPSC_HUMAN	SNF1_CANAL	SNF1_YEAST	SRK6_BRAOL	YNA3_CAEEL

Table 4.7: SwissProt Rel. 34 sequences containing the pattern “I.H.DLK.....D”. All 30 of these sequences are known or probable protein kinases.

database. In the description given below we assume the SwissProt annotation format.

The SwissProt database associates with most of its sequences annotations of sequence regions (the FT lines [8]). A typical region description looks like

FT DOMAIN 528 779 PROTEIN KINASE

where the keyword “FT” indicates that this is a region description line and the remaining line describes the region by giving its beginning and ending positions (from residue 528 up to and including residue 779 of the relevant database sequence) and its annotation (a protein kinase domain).

When presented with a pattern P we can use (as already mentioned) the offset list $L_D(P)$ to locate all the sequences in the database that match P . Assume that S is

such a sequence and that a substring that matches P begins at offset j within S . If P happens to fall in an annotated region of S (either entirely or in part) we can associate this region with P . Performing this process for every sequence S matching P results in a set $RS_D(P)$ of regions associated with P . Figure 4.11 gives an example of part of the output produced by our system for one of the three kinase patterns described above.

Given now a pattern P matched by a subsequence A of a query sequence Q , the question is how to use $RS_D(P)$ in order to characterize A . A number of approaches can be used. For example, if $RS_D(P)$ is large enough and the majority of its members agree in their functionality, then it can be inferred that A is quite likely to have the same functionality. Another consideration is the relative lengths of the pattern P and the regions described by the FT lines. If, for example, a pattern P has an extent of 15 residues while an annotated sequence region containing P has a length of 300 amino acids then one might not want to transfer the annotation of that region to P . In conclusion, the end user is expected to apply his/her expertise in deciding how to best exploit the information provided by the system.

Figure 4.12 describes two ways to use the sets $RS_D(P)$ in order to annotate regions of *YZ28_METJA*, thus extending the picture drawn in Figure 4.10.(b). The first approach (Figure 4.12.(b)) assigns an annotation X (e.g. X = transmembrane region) to a pattern P if (i) the size of $RS_D(P)$ is at least 15, (ii) the majority (80%) of the regions in $RS_D(P)$ are annotated as X and (iii) at least 50% of every region of $RS_D(P)$ annotated as X is covered by P . The second approach (Figure 4.12.(c)) shares the first two requirements above and relaxes the third by allowing the percentage of the annotated region covered by the pattern to be 30% or more.

Performance

The running time of a homology search for a query sequence Q depends on (i) the size of the set of patterns Π used and (ii) the actual number of local similarities (induced by the patterns matching Q) between Q and the database sequences. For the case of SwissProt Rel. 34 used here, typical searches for query proteins of size around a thousand residues take 4–6 seconds on a Pentium 266MHz computer with 256 MB of memory. It should be mentioned that the running time reported above is achieved by keeping all the program data (patterns and their offset lists) in memory. For SwissProt this data occupies around

200MB.

However, from preliminary work with the non-redundant protein database from NCBI it seems that although the search time components (i) and (ii) mentioned above are related to the size of the underlying database, this dependence is sublinear. In a sense, real protein databases induce a saturation of sorts on the size of Π : after a certain point, introducing new sequences in the database does not result in the generation of many new patterns. As a result, we expect that even as the size of the database used gets larger, the performance of the search phase (both running time per search and memory used) will increase at a much slower rate.

4.6 Discussion

We have presented here a methodology for performing sequence similarity searches based on the discovery of patterns over an underlying database D of proteins and the use of these patterns for the identification of homologies between a query sequence and the proteins of the database at hand. The crucial step for the success of the proposed approach is the collection of a set of patterns which is characteristic of the database D . In Chapter 3, we described a way to precisely define this set using statistical arguments. In this chapter we discussed how patterns provide more sensitivity in identifying significant homologies by introducing memory into the statistical computations. It was also shown how TEIRESIAS can be used to obtain the desired set of patterns. Finally, the utility of the methodology was exhibited using the SwissProt Rel. 34 database as a test-bed and showing how the system can be used for annotating query sequences. In this context we also discussed the potential of exploiting the discovered patterns in conjunction with the annotation of the underlying database towards characterizing even weak similarities between the query and the database sequences.

What sets the presented system apart from other pattern-based tools for homology detection (e.g. BLOCKS [44]) is the *completeness* of the set of patterns used. The patterns are discovered in an unsupervised manner from a very large training set, that of all the proteins within the underlying database D . There are no bias-creating prior assumptions on which sequences “should” be considered as members of the same family. As a result, the patterns discovered are expected to be more sensitive. Furthermore, by

considering together sequences of distinct functionalities we are able to discover weak similarities that span family boundaries (e.g. patterns that describe transmembrane regions). Such similarities, although not sufficient for the inference of functional annotations, give nevertheless useful information regarding the role of different parts of the query sequence under examination.

Another advantage of the system presented here is its performance. The speedup afforded by using patterns rather than scanning the entire database for every search is expected to become a factor as the size of genomic databases increases.

On the down size, the use of patterns is not guaranteed to find all existing local homologies in the sense that systems like BLAST or FASTA do. If there exists a sequence S within the database which is a singleton (in the sense that it has homologous regions with none or only a few other database proteins) then a query sequence Q homologous to S will go uncharacterized as there will be no patterns to associate it with S . It is expected, though, that such situations will become increasingly more rare as the size of the database D used gets larger.

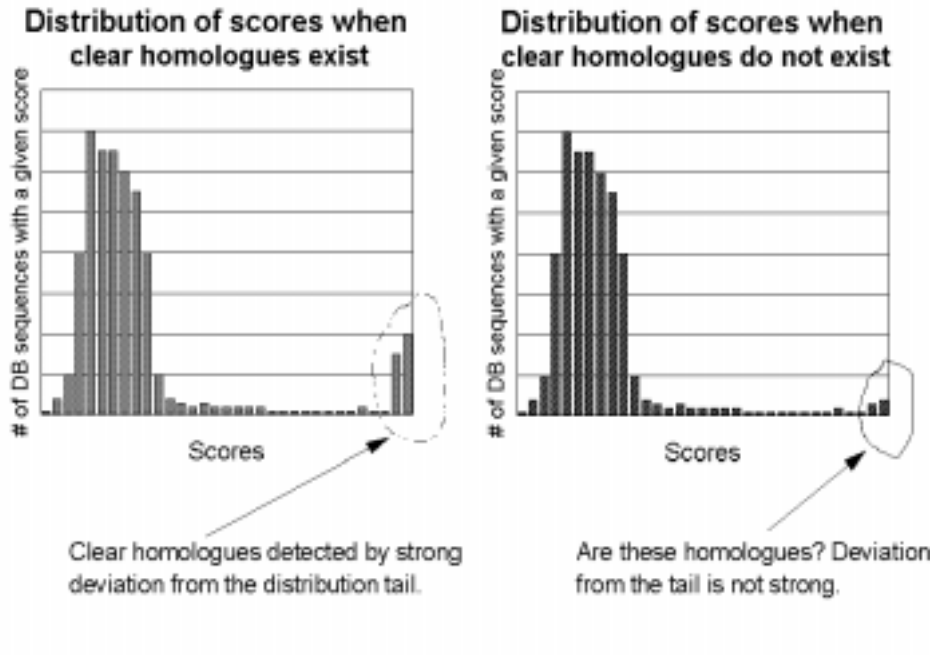


Figure 4.2: Example distribution of C_s scores for two FASTA queries. The leftmost graph corresponds to the situation where there are homologues of the query in the database. These homologues are detected as unexpectedly high scores at the tail of the distribution. The rightmost graph depicts a borderline case where the highest scores show a questionable deviation from the same tail.

```

s1 = . . . . .
s2 = . . . . .
s3 = . . . AMJFLLPKKLISMTRNLGTKHQWPFGLMPFWAA . . .
s4 = . . . ALJNMLPSHKISRAENPLFGLHWRFGLMEFGAS . . .
s5 = . . . DKJFLLPDKLIFMSRNLLEKHHLEFLNMPFRFD . . .
s6 = . . . AKJLMLPFHLIFKDRNKLEEHHPEFNRRERSGF . . .
s7 = . . . LGJNMLPLHRINLFENGPFEFHRPFMNRPDRDD . . .
s8 = . . . . .
s9 = . . . . .

```

Figure 4.3: The pattern “P.....N.....F” appears in five of the nine sequences shown here. The regions in the matching sequences around this *generator* pattern define a block. The length of the regions is chosen so as to maximize the block score (see text).

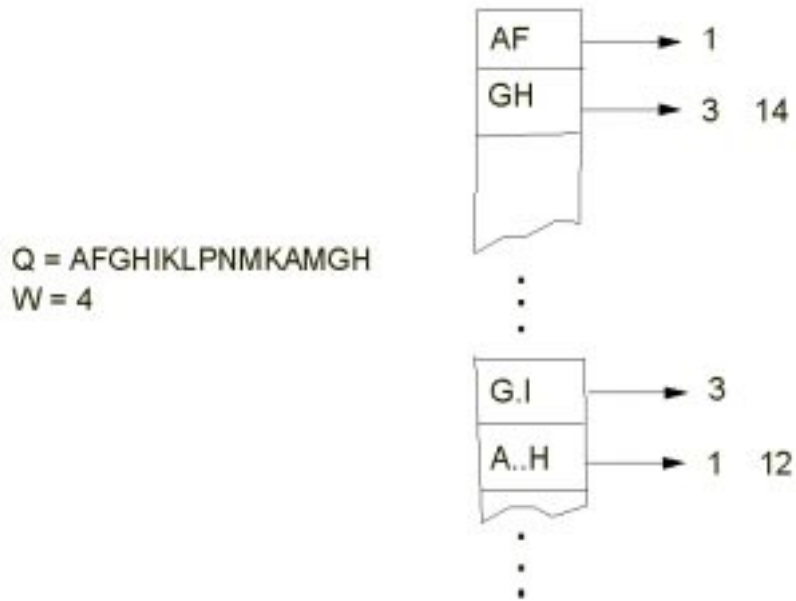


Figure 4.4: A snapshot of the hash table generated for the sequence $Q = \text{AFGHIKLPNMKAMGH}$. Instead of using actual numeric hash values to label the table entries we use a pattern, describing all the strings that hash to a particular hash value. Each hash entry points to a list of offsets. Every offset in that list marks the beginning of a substring in Q that matches the pattern labelling the hash entry at hand.

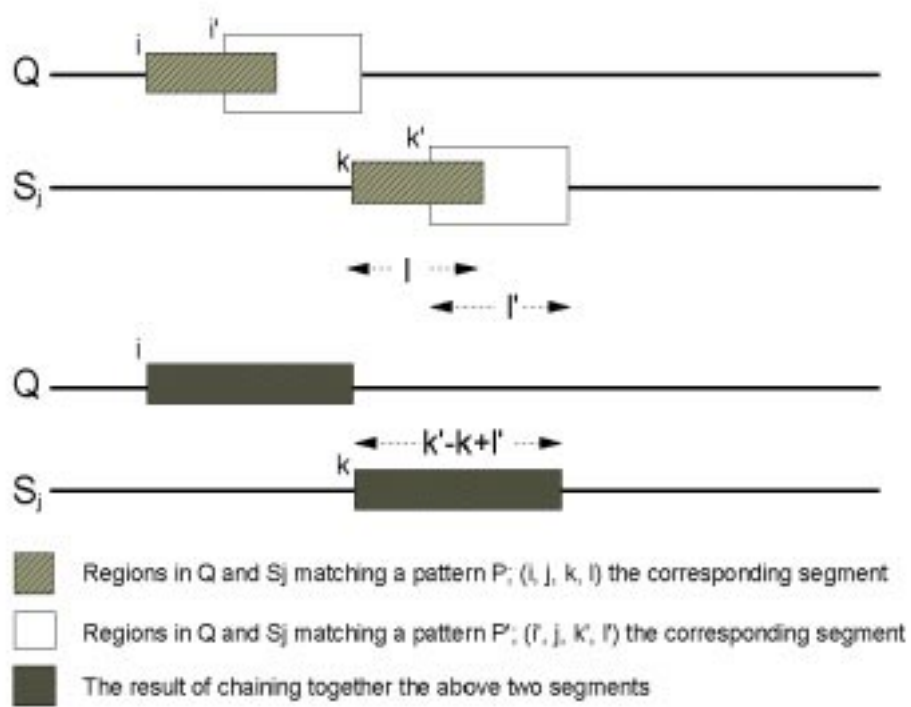


Figure 4.5: Each of the segments (i, j, k, l) and (i', j, k', l') indicates a local similarity between the query sequence Q and the data base sequence s_j . In the example shown here the two segments are compatible and they can be chained together into the single segment $(i, j, k, k' - k + l')$.

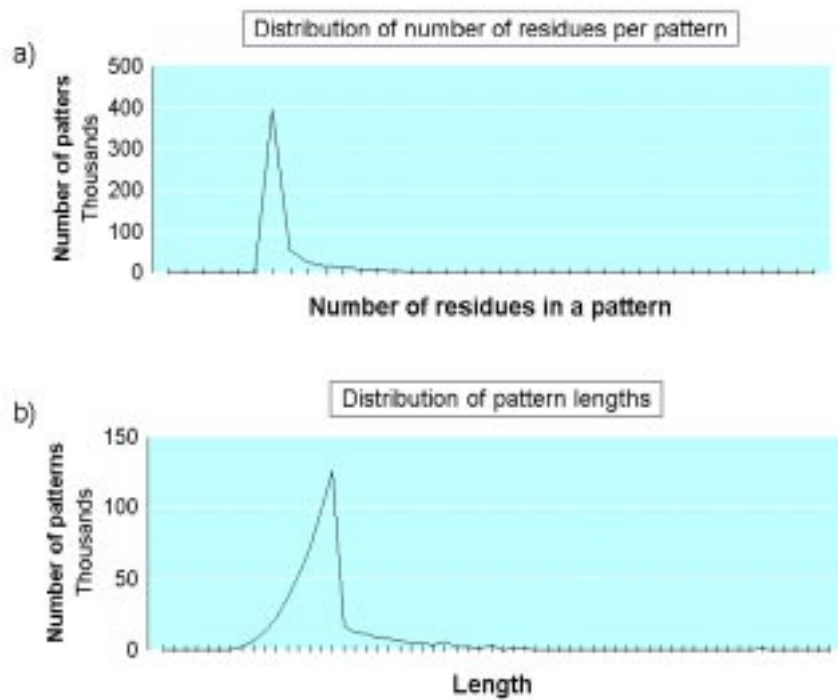


Figure 4.6: Distributions of (a) the number of residues and (b) the lengths of the SwissProt Rel. 34 patterns.


```

Score = 269
*****
Local Alignment(s) with the sequence H3_YEAST
Query   0      ARTKQTARKSTGGKAPRKQLATKAARKS
          ARTKQTARKSTGGKAPRKQLA+KAARKS
Seq     0      ARTKQTARKSTGGKAPRKQLASKAARKS
-----
Query   33     GVKKPHRYRPGTVALREIRRYQKSTELLIRKLPFQRLVREIAQDFKTDLRFQ
          GVKKPHRY+PGTVALREIRR+QKSTELLIRKLPFQRLVREIAQDFKTDLRFQ
Seq     33     GVKKPHRYKPGTVALREIRRFQKSTELLIRKLPFQRLVREIAQDFKTDLRFQ
-----
Query   96     EAYLVGLFEDTNLCAIHAKRVTIMPKDIQLARRIRGERA
          EAYLV LFEDTNL AIHAKRVTI  KDI LARR+RGER
Seq     96     EAYLVSLFEDTNLAAIHAKRVTIQKKDIKLARRLRGERS
-----

Score = 100
*****
Local Alignment(s) with the sequence CENA_HUMAN
Query   45     VALREIRRYQKSTELLIRKLPFQRLVR
          L+EIR+ QKST LLIRKLPF RL R
Seq     45     GWLKEIRKLQKSTHLLIRKLPFSRLAR
-----
Query   77     FKTDLRFQSSAVMALQEACEAYLVGLFEDTNLCAIHAK
          D  Q  A++ALQEA EA+LV LFED  L  +HA
Seq     79     RGVDFNWQAQALLALQEAAEAFVHLFEDAYLLTLHAG
-----
Query   111    IHAKRVTIMPKDIQLARRIRGERA
          +HA RVT+ PKD+QLARRIRG
Seq     113    LHAGRVTLFPKDVQLARRIRGLEE
-----

```

Figure 4.7: Local alignments of *H31_HUMAN* with a highly similar (*H3_YEAST*) and a moderately similar (*CENA_HUMAN*) protein. For every sequence, a number of local similarities are reported: in each case the relevant query (“Query”) and the data base sequence (“Seq”) regions are listed under one another with the resulting consensus regions between them. We use the character ‘+’ to indicate the alignment of chemically similar amino acids.

Score = 51

 Local Alignment(s) with the sequence NTNO_HUMAN

```

Query   196   VEKDILPHKVAFTGGGLRFILYPERPILEE
          E +   VA  G+GL FILYPE
Seq     369   HEHKVNIEDVATEGAGLVFILYPEAISTLS
-----

```

Score = 51

 Local Alignment(s) with the sequence NTNO_BOVIN

```

Query   196   VEKDILPHKVAFTGGGLRFILYPERPILEE
          E +   VA  G+GL FILYPE
Seq     367   HEHKVNIEDVATEGAGLVFILYPEAISTLS
-----

```

Score = 49

 Local Alignment(s) with the sequence KAPL_APLCA

```

Query   816   NAMIEMFKENYKLLKEYLETDIEVLKELDKNYK
          A  +F E  K LKEYLE+ +E   L
Seq     0     MAHNQVFPESQKWLKEYLESSLEQFENLFNKNV
-----

```

Figure 4.8: Top scoring alignments for the query sequence *YZ28_METJA*. The mutation matrix used is PAM 130.

Local alignments of sequence ----> YZ28_METJA along Y..S..I...DLK

Local alignment with the sequence ---> MP38_MOUSE

Score = 27

```
Query  24      DKYQINVSGIYNISDDILESDLKLHIAQLLFLI
          YQI      Y S DI+ DLK      +
Seq    129     LIYQILRGLKYYIHSADIIHRDLKPSNLAVNEDC
```

Local alignment with the sequence ---> MKK2_DROME

Score = 22

```
Query  24      DKYQINVSGIYNISDDILESDLKLHIAQLLFLI
          I      Y S DI  DLK
Seq    121     IMHEICAAVDYLHSRDIAHRDLKPENLLYTTTQ
```

Local alignment with the sequence ---> MP38_XENLA

Score = 22

```
Query  24      DKYQINVSGIYNISDDILESDLKLHIAQLLFLI
          YQI      Y S I+ DLK      +
Seq    130     LIYQILRGLKYYIHSAGIIHRDLKPSNLAVNEDC
```

Local alignment with the sequence ---> KRAF_CAEEL

Score = 20

```
Query  24      DKYQINVSGIYNISDDILESDLKLHIAQLLFLI
          Q+ +  Y S I+ DLK      L+ +
Seq    581     ILKQVSLGMNYLHASKNIIHRDLKTNNIFLMDDM
```

Figure 4.9: Top scoring local alignments for the query sequence *YZ28_METJA* induced by the pattern “Y..S..I...DLK”. The mutation matrix used is PAM 130.

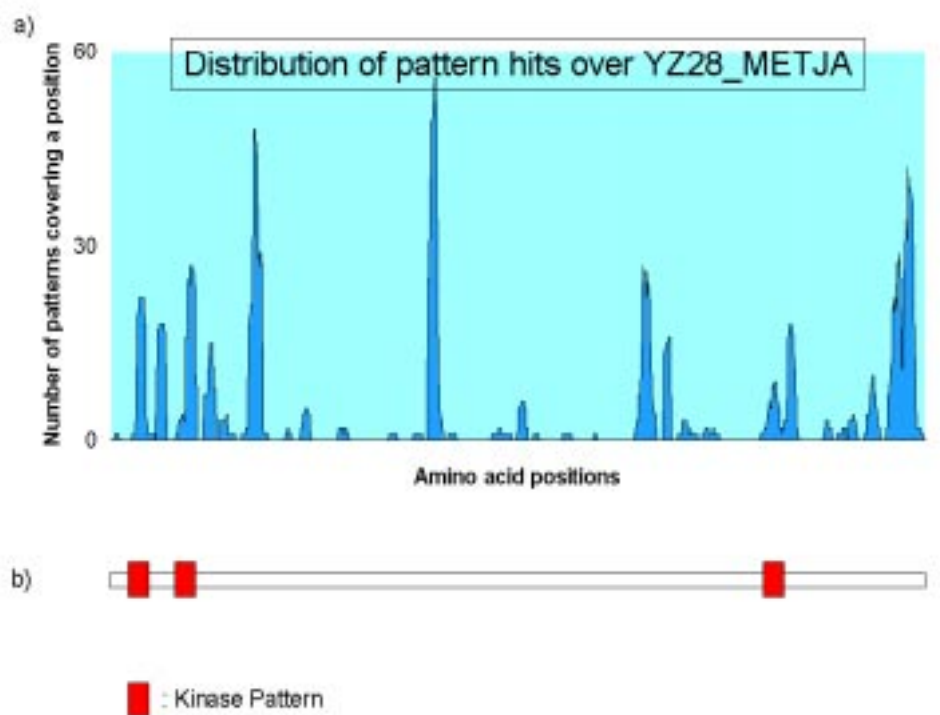


Figure 4.10: (a) There exist 410 patterns (among those discovered in the information gathering phase) matched by *YZ28_METJA*. A pattern “covers” a residue position if it starts before (or at) that position and ends after (or at) that position. The chart shows, for each residue position (x -axis), how many patterns (y -axis) cover that position. (b) The three kinase pattern discussed in the text match the sequence at offsets 35 (pattern “Y..S..I...DLK”), 112 (pattern “NIL.....IKL”) and 1052 (pattern “I.H.DLK.....D”). These offsets are depicted here relative to the spikes of the pattern distribution in (a).

```

#####
ID  ASK1_ARATH      STANDARD;      PRT;   363 AA.
DE  SERINE/THREONINE-PROTEIN KINASE ASK2 (EC 2.7.1.-).
-----> I.H.DLK.....D: Matching at offset 119
FT  DOMAIN         4      260      PROTEIN KINASE.
FT  ACT_SITE      123     123      BY SIMILARITY.

#####
ID  ASK2_ARATH      STANDARD;      PRT;   353 AA.
DE  SERINE/THREONINE-PROTEIN KINASE ASK2 (EC 2.7.1.-).
-----> I.H.DLK.....D: Matching at offset 119
FT  DOMAIN         4      260      PROTEIN KINASE.
FT  ACT_SITE      123     123      BY SIMILARITY.

#####
ID  CC2C_DROME      STANDARD;      PRT;   314 AA.
DE  CELL DIVISION CONTROL PROTEIN 2 COGNATE (EC 2.7.1.-).
-----> I.H.DLK.....D: Matching at offset 126
FT  DOMAIN         8      287      PROTEIN KINASE.
FT  ACT_SITE      130     130      BY SIMILARITY.

#####
ID  CC2_DICDI       STANDARD;      PRT;   296 AA.
DE  CELL DIVISION CONTROL PROTEIN 2 HOMOLOG (EC 2.7.1.-) (P34 PROTEIN
DE  KINASE).
-----> I.H.DLK.....D: Matching at offset 125
FT  DOMAIN        10      288      PROTEIN KINASE.
FT  ACT_SITE      129     129      BY SIMILARITY.

#####
ID  CC2_SCHPO       STANDARD;      PRT;   297 AA.
DE  CELL DIVISION CONTROL PROTEIN 2 (EC 2.7.1.-) (P34 PROTEIN KINASE).
-----> I.H.DLK.....D: Matching at offset 130
FT  DOMAIN         4      293      PROTEIN KINASE.
FT  ACT_SITE      134     134      BY SIMILARITY.

```

Figure 4.11: Analyzing individual patterns using the SwissProt annotation: some of the data base sequences matching the pattern “I.H.DLK.....D”. For every such sequence its ID and DE lines are reported [8], giving the SwissProt name of the sequence and a short description of its functionality. Next follows the offset within the sequence where the match originates. Finally, there are the FT lines for all the annotated regions having an intersection with the region covered by the pattern.

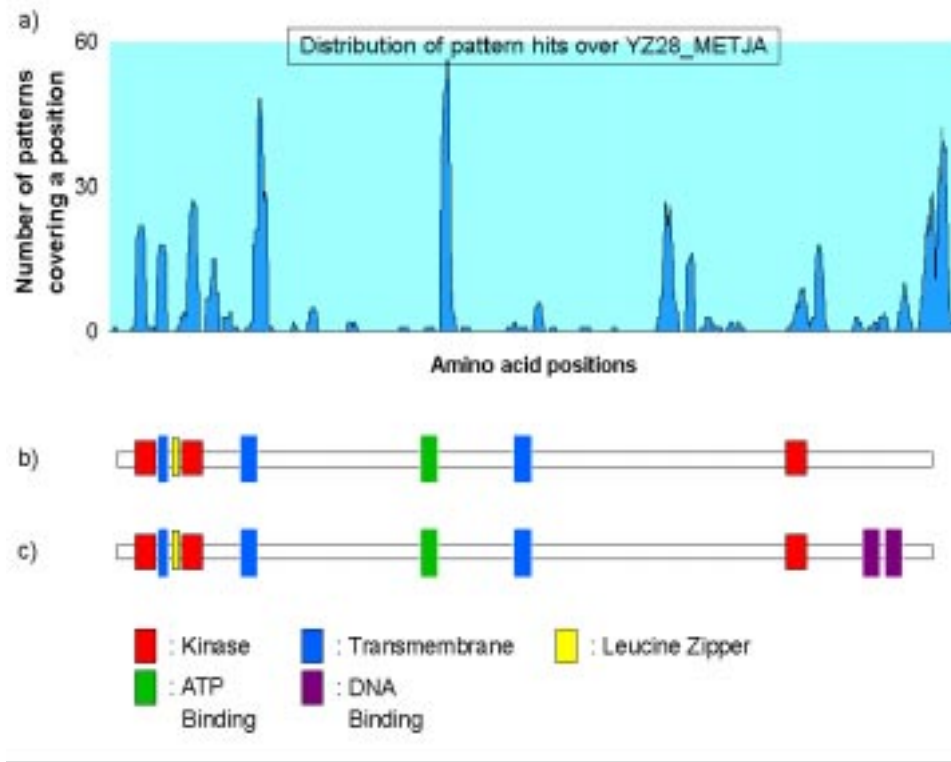


Figure 4.12: Characterization of various segments of *YZ28_METJA* from the annotation of the patterns matched by these segments. The annotation of the patterns is obtained by exploiting the information available for the various regions of the database sequences also matching these patterns. The segments are shown again relative to the spikes of the distribution of patterns over the entire *YZ28_METJA*.

Chapter 5

Other Applications And Future Work

Pattern discovery can be a central element to many applications, both in Biology and other fields. In this chapter we briefly examine a few such applications. From the area of Biology we concentrate on the problem of *multiple sequence alignment*. A second application comes from the field of computer security and involves the identification of network-based security attacks. For these two problems we discuss the way in which TEIRESIAS has been instrumental in the proposed solutions.

The chapter concludes with an outline of future research plans. These plans include (i) improvements and extensions in the algorithm, and (ii) exploitation of the seqlets in the protein folding problem.

5.1 Multiple Sequence Alignment

The multiple sequence alignment (MSA) problem was informally introduced in Section . The input to the problem is a set of biological sequences (usually proteins). The objective is to align these sequences (using a predefined set of allowable edit operations) so as to bring out the commonalities among them. More precisely, given a set $S = \{s_1, s_2, \dots, s_N\}$ of sequences from an alphabet Σ , a multiple sequence alignment of this set is defined as an $N \times M$ table \mathcal{A} , where $M \geq \max\{|s_i|\}$. The entries of this table are characters from

the set $\Sigma \cup \{-\}$, where ‘-’ is a special character (indicating a gap) not in Σ . The i -th row of the table corresponds to the sequence s_i and describes how it is aligned relative to the remaining sequences. In particular, for each $1 \leq i \leq n$ and for every $1 \leq j < |s_i|$ there exist j_1 and j_2 with $1 \leq j_1 < j_2 \leq M$ such that

- $\mathcal{A}[i, j_1] = s_i[j]$,
- $\mathcal{A}[i, j_2] = s_i[j + 1]$, and
- $\forall k, j_1 < k < j_2 : \mathcal{A}[i, k] = -$.

In other words, the i -th row of \mathcal{A} is generated from sequence s_i , by inserting zero or more gaps (i.e. ‘-’ characters) between consecutive characters of s_i .

In practice, the term “multiple sequence alignment” is used to refer to the optimization version of MSA. In this case a cost is assigned to every possible alignment (i.e. every matrix \mathcal{A}) and one is looking for the alignment with the maximum cost. The most often used cost functions work by examining each column of the alignment \mathcal{A} under consideration and by penalizing mismatches and gaps while rewarding matching (or related) residues [1, 110].

We have already discussed a number of algorithms for the MSA problem (see Section 5.1). The majority of them builds an (approximation to the) optimal alignment in a bottom-up manner, starting from the pair-wise alignment of all possible sequence pairs. There are, however, algorithms that follow an alternative approach: they begin by identifying local similarities shared by several input sequences and then use these similarities as anchor points around which the global alignment is built. One example of this approach [98] was discussed in Section 2.2.2. In that particular case, the first step was the identification of substrings that belong to *all* the input sequences. This requirement (i.e. membership to all input sequences) makes it relatively easy to “chain” these strings together into a larger alignment. It also makes the algorithm rather useless as there are very few input sets that would abide to such a restriction.

Consequently, an extension of this approach is almost imperative. In particular, one must allow for local similarities that may be shared by only some of the input sequences. MUSCA (see reference in Section 1.6) is a multiple sequence algorithm that does exactly that: it uses motifs discovered by TEIRESIAS and puts them together in order to build a

global alignment of the input sequences. In particular, consider any alignment \mathcal{A} of the input set $S = \{s_1, s_2, \dots, s_N\}$ and for any column j of the table \mathcal{A} define the following function:

$$H_K(j) = \begin{cases} l, & \text{if } l \geq K \text{ and } A[i_1, j] = A[i_2, j] = \dots = A[i_l, j] = \sigma, \\ & \text{for some } \sigma \in \Sigma \text{ and distinct } i_1, i_2, \dots, i_l. \\ 0, & \text{otherwise} \end{cases}$$

where K is an arbitrary positive integer such that $K \leq N$.

The optimization problem addressed by MUSCA is the following variant of the MSA problem.

The K -MSA Problem

Input: A set $S = \{s_1, s_2, \dots, s_N\}$ of sequences and a positive integer $K \leq N$.

Output: An multiple sequence alignment \mathcal{A} that maximizes the following expression:

$$\max_{\mathcal{A}} \left(\sum_{\text{column } j} H_K(j) \right). \quad (5.1)$$

In other words, the K -MSA problem (i) imposes the (biologically relevant) restriction that alignment columns that “count” must have at least K identical characters, and (ii) looks for an alignment that maximizes the number of aligned characters that agree. The problem, as defined above, can be shown to be MAX-SNP hard. MUSCA provides an approximation solution. It works by first employing **TEIRESIAS** to find patterns with support at least K and then by combining these patterns into a global alignment. In particular, the intention is to subselect (among all patterns discovered by **TEIRESIAS**) a subset that admits a *feasible ordering* leading to a near-optimal alignment.

Definition 5.1.1 (*Feasible ordering*) An ordering of patterns $\mathcal{R} = (P_1, P_2, \dots, P_m)$, where each pattern P_i has support at least K in S , is said to be feasible for S if there exists a multiple sequence alignment \mathcal{A} such that:

1. $\forall P \in R, \exists j_P, \forall (x, y) \in L_D(P)$:

$$\begin{aligned} \mathcal{A}[x, j_P] &= s_x[y], \\ \mathcal{A}[x, j_P + 1] &= s_x[y + 1], \\ &\dots \\ \mathcal{A}[x, j_P + |P| - 1] &= s_x[y + |P| - 1]. \end{aligned}$$

2. $\forall r, r'$ with $1 \leq r < r' \leq m : j_{P_r} \leq j_{P_{r'}}$.

Allowing for patterns that can be shared by only some of the input sequences introduces a number of non-trivial problems in building a K -MSA. These problems originate from potential conflicts between the discovered patterns. Figure 5.1 gives an example of such a situation. Because of such problems, it becomes imperative to characterize those sets of patterns which can give rise to valid alignments. In what follows, when referring to a pattern we always mean one that has support at least K in the input set of sequences S .

Definition 5.1.2 (*Feasible pair*) *Two patterns P_1 and P_2 are pairwise feasible if and only if*

1. *For all input sequences matching both P_1 and P_2 , the patterns always appear in the same order (i.e. either P_1 always before P_2 or P_2 before P_1).*
2. *In all sequences in which the matches of P_1, P_2 are overlapping, the amount of the overlap is constant. More formally, there is an integer d such that for all offset pairs of the form $((x, y_1), (x, y_2))$ where $(x, y_1) \in L_D(P_1)$ and $(x, y_2) \in L_D(P_2)$:*

$$y_2 - y_1 = d.$$

Definition 5.1.3 (*Consistent ordering*) *An ordered set of patterns $R = (P_1, P_2, \dots, P_m)$ is said to be consistent with the input set S if and only if whenever two or more patterns from R appear in some sequence of S , then the order of their matching offsets is the same as the relative order of the patterns in R .*

It might seem as if a consistent ordering of a set of patterns could be sufficient to guarantee feasibility. This is not the case, though, as Figure 5.2 depicts.

Definition 5.1.4 (*Domain crossing*) Given an ordered set of patterns $R = (P_1, P_2, \dots, P_m)$ a domain crossing error is said to occur when R is consistent but not feasible. ■

It is not difficult to show now that a given set of patterns Π has a feasible ordering if and only if none of the following is true:

1. There exists in Π a pair of patterns which are pairwise infeasible.
2. There exists a subset of Π admitting no consistent ordering.
3. There exists a subset of Π for which all consistent orderings introduce domain crossing errors.

A set of patterns for which no feasible ordering exists, is called *infeasible*. Interestingly, every infeasible set can be grouped into subsets (not necessarily disjoint) so that every such subset violates *exactly one* of the above three properties. These subsets are called *basic infeasible sets*. Furthermore, it can be shown that removing any member-pattern from a basic infeasible set renders the set feasible. MUSCA approximates an optimal solution to the K -MSA problem by first locating all the basic infeasible sets in the output of TEIRESIAS and then removing the *minimum* number of patterns so that all the basic infeasible sets become feasible.

The detection of the basic infeasible sets can be done efficiently, using a graph formulation of the problem. In particular, consider a directed graph $G = (V, E)$ where

$$V = \{u_P \mid P \text{ is a pattern produced by TEIRESIAS}\}.$$

An edge is placed between the vertices u_P and u_Q if there are input sequences matching both P and Q and in all such sequences the matching offset for P occurs before the matching offset for Q . Every such edge has one of three possible labels. The possibilities are (i) **forbidden** if P, Q are pairwise infeasible, (ii) **overlap** if P, Q overlap, or (iii) **nonoverlap** if P, Q do not overlap. A vertex-disjoint path in this graph is called *valid* if it contains no edges labelled **forbidden**. If, furthermore, the path contains only edges labelled **overlap** then it is called *overlap-only* path.

In order to use the graph G for the detection of domain crossing errors, a weight $D(u_P, u_Q)$ is assigned to every edge $u_P \rightarrow u_Q$ that is not labelled **forbidden**. This

weight is defined to be the minimum distance between between the offsets of successive occurrences of the patterns P and Q over all the sequences matching both these patterns. Furthermore, the weight $\mathcal{D}_{\mathcal{H}}$ of a valid path \mathcal{H} in G is the sum of the weights of its constituent edges. Finally, the graph G is said to be *consistent with respect to a vertex* u_P if for every vertex u_Q and for all pairs of vertex-disjoint paths $\mathcal{H}_1, \mathcal{H}_2$ leading from u_P to u_Q :

- $\mathcal{D}_{\mathcal{H}_1} = \mathcal{D}_{\mathcal{H}_2}$, if both \mathcal{H}_1 and \mathcal{H}_2 are overlap-only paths,
- $\mathcal{D}_{\mathcal{H}_1} \geq \mathcal{D}_{\mathcal{H}_2}$, if \mathcal{H}_1 is an overlap-only path while \mathcal{H}_2 is not.

Having this graph in place, it is not difficult to prove that the basic infeasible sets are the following (notice that for the remaining of this section we identify the verices of G with the patterns that they represent):

1. All sets F_1, F_2, \dots, F_{N_f} containing pairs of vertices connected with an edge labeled **forbidden**.
2. All sets C_1, C_2, \dots, C_{n_c} where each C_i defines a cycle in the graph G .
3. All sets P_1, P_2, \dots, P_{n_p} , where every P_i contains the vertices that form a closed path in G .

Standard graph traversal algorithms can be used to compute these sets. Let us assume that all the basic infeasible sets have been found. As mentioned earlier, removing one vertex from every such set leaves us with a set of patterns which admit a feasible ordering. With this fact in mind, MUSCA will try to identify (and then remove) the smallest possible vertex set $U \subset V$ with the property that each basic infeasible set contains at least one vertex from U .

The task is achieved by observing that the problem which we are trying to solve is an instance of the *set-covering* problem. In set-covering one is given a pair of sets (D, B) where $B \subseteq 2^D$ and every element of D is contained in at least one set in B . The question then is to find the minimum number of sets in B whose union is D . In our case,

$$D = \{F_1, \dots, F_{n_f}, C_1, \dots, C_{n_c}, P_1, \dots, P_{n_p}\}$$

and the set $B = \{U_1, U_2, \dots, U_n\}$ contains one element for every vertex $u_i \in ((\cup_i F_i) \cup (\cup_i C_i) \cup (\cup_i P_i))$. In particular,

$$U_i = \{F_j \mid u_i \in F_j\} \cup \{C_j \mid u_i \in C_j\} \cup \{P_j \mid u_i \in P_j\}.$$

Using this modeling and any approximation algorithm for the set-covering problem, we can obtain a feasible set of patterns. There, is though, one detail that has not been addressed: the resulting pattern set ought to give rise to a near optimal alignment. To accommodate this requirement the formulation of the set-covering problem has to be modified. More specifically, every set $U_i \in B$ is assigned a weight $w(U_i)$ that mirrors the importance of the corresponding pattern $u_i \in V$ relevant to the underlying cost function given in equation 5.1. In particular, if the pattern u_i has r regular character and support k in the input set S , then $w(U_i) = rk$. With this formulation, the problem to solve becomes a weighted variant of the set-covering problem: find a subset B' of B such that $\sum_{U_i \in B'} w(U_i)$ is as small as possible and so that $\cup_{U_i \in B'} U_i = D$.

An approximation algorithm to this weighted set-covering problem is given in [22]. The approximation factor is $1 + |D|$. Employing this algorithm we can compute a set of patterns whose removal from the original collection of patterns produced by **TEIRESIAS** results in a set admitting a feasible ordering. The alignment \mathcal{A} induced by this latter set can then be easily constructed. Further details along with multiple sequence alignments produced by **MUSCA** on real sequences can be found in the relevant reference given in Section 1.6

5.2 Computer Security

Pattern discovery algorithms find applications in many domains other than Biology (in different contexts they are referred to as either *data* or *knowledge mining* algorithms). Realizing this, we strived right from the start to design **TEIRESIAS** so as to be a domain-independent tool. Our intention was to have an algorithm that could be used for the discovery of patterns in every field where the data can be modeled as strings over a finite alphabet. In this section we briefly present an application of **TEIRESIAS** in the area of computer security and more specifically in the construction of a detection intrusion system for network-based utilities such as *ftp* or *mail*. In particular, we are interested in

identifying suspicious looking uses of these utilities which usually indicate attempts to exploit security loopholes in these programs. The research described here was performed in cooperation with the Global Security Analysis Group of IBM Research in Zurich and the resulting system is intended to become part of AIX, IBM's version of UNIX. In a nutshell, the system uses **TEIRESIAS** to discover patterns in audit trails generated by normal uses of the utilities. The patterns are subsequently used for the detection of intrusion attempts against the system: sessions that deviate consistently from the ordinary behavior prescribed by these patterns are flagged as potential attacks.

Generally, an intrusion detection system dynamically monitors actions that are taken in a given environment and decides whether these actions are symptomatic of an attack or constitute a legitimate use of the environment. Essentially, two main intrusion detection methods have been proposed. The first method uses the knowledge accumulated about attacks and looks for evidence of their exploitation; this method is referred to as *knowledge-based*. The second method builds a reference model of the usual behavior of the system being monitored and looks for deviations from the observed usage; this method is referred to as *behavior-based*.

In the knowledge-based approach, the underlying assumption is that the system knows *all* possible attacks. There is some kind of a signature for each attack and the intrusion detection system searches for these signatures when monitoring the traffic. E.g., one may monitor the audit trails on a given machine, the packets going onto the net, etc. An advantage of this method is that no or only few false alarms are generated, i.e. the false alarm rate is low; the main disadvantage is that only those attacks which are already known can be located. Any newly developed intrusion attack would usually remain undetected since its signature is still unknown and thus the system will never search for it.

Unfortunately, there are nowadays so many attacks that the set of signatures is growing very fast. Also, some signatures are difficult to express and an algorithm to search for them can be rather time-consuming. Nevertheless, this approach has proven its usefulness and there are products using this approach available on the market (e.g. NetRanger by Cisco Systems, Inc., and RealSecure by Internet Security Systems, Inc.).

The behavior-based approach starts from the assumption that if an attack is carried out against a system, its "behavior" will change. One can thus define a kind of normal

profile for a system and watch for any deviation from this defined normal profile. Different techniques can be applied (e.g. statistics, rule-based systems, neural networks) using different targets (e.g. the users of the system, the performances of the network, the CPU cycles, etc...). The main advantage of this method over the knowledge-based one is that the attacks do not need to be known in advance, i.e. that unknown attacks can be detected. Thus, the detection remains up-to-date without having to update some database of known signatures. But there are disadvantages: deviations can occur without any attack (e.g. changes in the activity of the user, new software installed, new machines, new users, etc.). Therefore, all known efforts in this direction have been facing a rather high rate of false alarms.

An example of a behavior-based intrusion detection system is given in [32]. It is described therein how to model the behavior of the “sendmail” daemon (a program running permanently in the background without user interaction), using the sequences of system calls that this program generates while running. The idea is to build a table of all the sequences of a given *fixed* length (here 5, 6, and 11) of consecutive system calls that could be found when watching such a *sendmail* daemon running. The idea is that if one tries to take advantage of a vulnerability in the *sendmail* code, then this would generate a sequence of systems calls not found in a “normal” table, i.e. a table generated from samples with normal behavior. However, when experimenting with this approach, one discovers that the table necessary can become fairly large. It must also be stressed that all the sequences of system calls in this table have the same length, i.e. lengths of 5, 6, and 11. As it has been shown [26], though, when trying to find what the best length for the sequences is (“best” meaning producing the shortest table of patterns while covering all possible sequences) the result is that the “best” length is 1. This means that the system does not search for unseen sequences but for unseen system calls. The consequence is that if an attack does not use any unseen system call it will not be detected. This is generally unacceptable since it may be possible to run an attack without using a previously unseen system call.

Using fixed length patterns to model the behavior of a system utility is not always a good idea. By monitoring, for example, the audit trail generated by the *ftp* daemon in UNIX one finds several long patterns, each with a different length, that appear quite often. Since the described fixed-length approach does not consider such a characteristic,

any result of an intrusion detection method based on a fixed-length approach is distorted and certain intrusions and/or misuses cannot be detected.

So, it seems that one has to consider patterns of *variable* length. Such an approach is examined in [62] and the results obtained are shown therein to be very promising. However, the patterns presented in that work are constructed manually due to the lack of an automated method. It is obvious that such a manual selection or design of the patterns is inadequate for an automatic intrusion detection of the kind considered here.

In our work, we extend the approach of [32] by introducing the automatic discovery of variable-length patterns. Our approach works in two distinct phases: an off-line *training* phase and an on-line *operation* phase. During training, trails of normal sessions are presented as input to the pattern discovery phase. The resulting patterns are placed in a *pattern table*. Patterns from this table are used in the operation phase, in order to match the audit trail of an ongoing session. If there are large parts of the session that cannot be matched by these patterns, then this is considered to be a serious deviation from the normal behavior. In such cases alarms are raised, notifying the operating system of a potential attack.

The particular features of **TEIRESIAS** make the algorithm especially appropriate for the training phase. More specifically, **TEIRESIAS** can discover patterns that are maximal and of arbitrary length. This is important since a table of long patterns appears to be more “representative” of a specific process than a table of short patterns. Since longer patterns usually contain more context information, it appears that they are more significant for a process than short patterns. On the other hand, short patterns are not necessarily unique for a specific process, but may appear in other processes. It is even possible that short patterns are part of an attack. The longer a pattern is, the lower the probability that this pattern is part of other processes or of an attack.

Another advantage of long patterns is that the pattern table resulting from the training phase has a relatively short length. This, in return, allows the pattern matching process (that takes place during the operation phase) to proceed faster. Speed is an important consideration since the envisioned use of our approach is in a real-time environment, within the context of an operating system.

The general architecture of the proposed system is depicted in Figure 5.3. During the training phase a number of normal sessions (denoted in the figure as “Model Process”)

are used for the characterization of normal behavior. The audit trail of each such session (which is either the sequence of system calls that the session generates or any other set of event logs allowed by the operating system) adds one sequence in the input to **TEIRESIAS**. The set that **TEIRESIAS** will process contains then as many sequences as are the sessions used during the training phase.

A single item in an audit trail usually contains a number of fields such as the process name, the process id, the number of the system call etc. Most of this information is not relevant for the pattern discovery phase and can be discarded. This task is performed by the *Translation* module which consistently translates every audit trail item into a character from a discrete alphabet. The translation is bijective, i.e. identical events are translated into the same character and a character is used to code for a single event type. The mapping between the alphabet characters and the audit trail events occurs on the fly and is recorded into the *Translation Table* for use in the operation phase. Finally, before the translated audit trails are passed to **TEIRESIAS** for processing they are cleaned-up: this is the function of the *Reduction* module. This module performs two types of transformations:

- Duplicate sequences are removed.
- Consecutive occurrences of the same character are collapsed into a smaller number of the same character.

The first task results in a set of unique strings. Duplicate strings do not add any value to the further processing. The second task removes from the input semantically uninteresting repetitions of the same event. Such repetitions are not unusual when dealing with system utilities. For example, the *ftp login* session has to close several file handles inherited from the *inetd* process. Since the *inetd* process is not always in the same state, the number of its file handles may vary. Closing all the unneeded file handles results therefore in a varying number of file close operations. In general, it has been observed that subsequences of $N, N > 1$, identical events are quite frequent, with N exhibiting small variations.

There are two possible ways to collapse characters:

- The identical consecutive characters are replaced with an extra, not yet used character.

- The N identical consecutive characters are condensed into $M, 1 \leq M \leq N$, characters.

The first approach increases the number of unique events and possibly also the number of patterns. Since the number of patterns should be kept small, the second approach with $M = 1$ has been selected. The newly created strings have reduced semantics compared to the original ones, but no case is known where the character aggregation impacts the operation of the intrusion detection system.

Finally, the cleaned-up set of input sequences are passed to **TEIRESIAS** for processing. The parameter settings for **TEIRESIAS** are usually quite conservative: for *ftp* we choose the minimum support to be 2 while L, W are set to 6 and 8 respectively. It turns out that because of the nature of the data, a smaller ratio of L/W offers no noticeable advantage. This observation has been substantiated through experimentation with many different settings for L and W . The patterns resulted from the processing of the input are then stored in the *Pattern Table*.

The operation phase of the system monitors in real time sessions of the utility (or utilities) that were modeled during the training phase. As with the model processes, an *Actual Process* generates an audit trail which has to be first translated (using the mapping stored in the *Translation Table*) and then reduced. The resulting string is then passed to the *Pattern Matching* module which, in an on-line fashion, tries to match the part of the input sequence seen thus far with the patterns in the *Pattern Table*. When this module identifies on the input sequence a substring of length n that cannot be covered by the patterns of the *Pattern Table*, then an alarm is raised, flagging a potential attack. The parameter n is for the time being decided experimentally, by trying several different values and checking which one results in the most reliable alarm raising. For *ftp*, for example, we use $n = 6$.

5.3 Future Work

We conclude this chapter with suggestions for future work. There are two main directions to follow. One has to do with improvements in the algorithm and the second with the exploitation of the seqlet sets for the extraction of quality information from the relevant databases.

5.3.1 Improvements in TEIRESIAS

One of the criticisms about TEIRESIAS is that it is not flexible enough in its treatment of *related* amino acids. Recall from Section 1.4.2 that amino acids can be grouped according to their chemical properties. This grouping mirrors the tendency of amino acids to mutate with higher probability within their respective groups rather than out of them (in PAM and BLOSUM matrices this tendency is recorded as higher values for the entries that correspond to pairs of same-group amino acids). Other types of groupings are also possible [41].

The most common way to represent such inter-amino acids relationships in patterns is through the use of the [XYZ] notation, indicating that a given position of the pattern can be matched by more than one residues (in this example, by residues X, Y or Z). The user only has to specify the allowable groupings of amino acids, i.e. which amino acids can appear together in a bracket. There are several algorithms [57, 77, 79, 89, 96] which can handle this type of pattern.

It is possible to extend TEIRESIAS in order to allow for *ambiguous* matches involving brackets. There are two cases to consider. First, when the amino acid groups are disjoint, i.e. every amino acid belongs to at most one group (this is the case with the classification shown in Table 1.1. In this particular situation no change to TEIRESIAS is necessary: all that needs to be done is rewrite the input using a new character for each amino acid group (Figure 5.4) and then use TEIRESIAS “as is” for processing this modified input.

A second, more complicated case arises when residues can belong to more than one groups. Table 5.1 shows such an example. One way for TEIRESIAS to allow bracketed expressions based on such multi-property groupings is to exploit existing tools. It is, for example, possible to “attach” *Emotif* [79] as a back end to TEIRESIAS. Emotif can then process every pattern produced by TEIRESIAS replacing where it is appropriate bracketed expressions for don’t-cares. This approach works fine as long as one accepts the restriction that a pattern must have at least L (the TEIRESIAS parameter) regular characters: by construction, every pattern produced by TEIRESIAS has this property. Since this might not always be acceptable, we discuss below a way to extend TEIRESIAS so that the discovery of arbitrary bracketed expressions becomes an integral part of the algorithm. Such an extension also has the advantage of being more efficient compared

to schemes that post process the output of TEIRESIAS.

A, G	Small
S, T	Small hydroxyl
K, R	Basic
F, Y, W	Aromatics
H, K, R	Basic
I, L, V	Small hydrophobic
I, L, V, M	medium hydrophobic
E, D, N, Q	Acidic/amid
A, G, P, S, T	Small Polar

Table 5.1: Grouping of amino acids according to a number of chemical properties.

It is possible to handle bracketed expressions without major changes to the algorithm design. What has to be done, is just use an extended alphabet

$$\Sigma' = \Sigma \cup \{r_1, r_2, \dots, r_m\}$$

where r_i are distinct characters not in Σ and there is one such character for each of the sets R_1, R_2, \dots, R_m ($R_i \subset \Sigma$) that define the allowable residue groupings. When building the elementary patterns now, a character $\sigma \in \Sigma$ also contributes an offset to every r_i for which the corresponding set R_i contains σ . After the generation of the elementary patterns the convolutions are performed as before, using now the alphabet Σ' . Complications, however, can arise: one such situation is depicted in Figure 5.5. After convolving two patterns into a larger one, it may be necessary to replace in the resulting pattern a character $r_i \in (\Sigma' - \Sigma)$ with another character $r_j \in \Sigma'$, where $R_j \subset R_i$. In order to achieve this goal we must redefine the prefix-wise less and suffix-wise less relationships presented in Section 2.3.3. In particular, we must take into account set-inclusion relationships between the R_i . We are currently in the process of implementing a bracketed version of TEIRESIAS that can handle arbitrary groupings, such as those described above. Results and a detailed description will be presented in an upcoming publication.

5.3.2 Validation of Seqlets

In Chapter 3, we introduced the notion that the set of seqlets obtained from the processing of large and diverse databases has the potential of becoming a sort of vocabulary of life. This claim is based on the assumption that statistical significance (which is the only criterion that we use when choosing a pattern for becoming a member of the seqlet set under construction) is sufficient — if not always, at least in the majority of cases — for inferring biological significance. We also discussed why a full validation of this assumption would require checking the biological function (i.e. assigning semantics) of each and every seqlet.

Given the big number of seqlets that we find and the substantial effort that is required in order to assign function to every seqlet using wet-lab methods, we tried to use alternative approaches for evaluating the quality of the discovered seqlets. One such approach is the exploitation of existing annotation, as described in Section 3.2.2 and depicted in Table 3.5. The annotation that we used to obtain results such as those shown in Table 3.5 is of a functional nature: it is describing the function of entire proteins or protein domains. Useful as functional information may be, it nevertheless has its limitations. For example, it is not possible to identify the importance of a seqlet if that seqlet only describes structural features (e.g. an α -helix) and not a self-contained functionality.

Fortunately, repositories of (potentially exploitable) structural information are also available. One such repository is the Protein Data Bank (PDB) described in Section 1.5. PDB is a database of protein structures. We plan to use the information offered by PDB in order to structurally characterize seqlets.

Our approach has the following steps:

- Identify all seqlets that are matched by at least two sequences in PDB.
- For each such seqlet P , find all regions of PDB proteins that match P and put these regions in the set T_P . Since we require P to be matched by at least two PDB proteins it follows that $|T_P| \geq 2$.
- For every region $r \in T_P$ extract its backbone coordinates from PDB. Create an optimal multiple structural alignment of the backbones of all the regions of T_P .

The structural alignment of 3-dimensional objects entails the application of rigid transformation for superimposing one object on top of another so as to minimize an objective function. In the case of backbones, the function that we try to minimize is the *root mean square* error or *RMS* error of the multiple alignment (to be defined shortly).

- Assign to each seqlet P a value $v(P)$ equal to the RMS error of the optimal alignment of T_P .

RMS errors are used in crystallography as standard ways of evaluating the structural “closeness” of chemical compounds. Consider two *aligned* 3D protein fragments A and B of the same length which are expressed as atom sequences (in the N-terminal to C-terminal direction). More specifically, let

$$A = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_n)$$

and

$$B = (\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_n)$$

where $\mathbf{a}_i = (x_i, y_i, z_i)$ gives the coordinates of the i -th atom in A and $\mathbf{b}_i = (x'_i, y'_i, z'_i)$ provides similar information for the i -th atom of B (in real situations, only the backbone atoms of the two compared polypeptides are considered). The RMS error of the alignment is defined as

$$RMS(A, B) = \frac{1}{n} \sqrt{\sum_{i=1}^n |\mathbf{a}_i - \mathbf{b}_i|^2} = \frac{1}{n} \sqrt{\sum_{i=1}^n ((x_i - x'_i)^2 + (y_i - y'_i)^2 + (z_i - z'_i)^2)}.$$

When dealing with many fragments there are several different ways to define the collective RMS error of the alignment. We are using the following

$$RMS(A_1, A_2, \dots, A_N) = \max_{i \neq j} \{RMS(A_i, A_j)\},$$

where all the A_1, A_2, \dots, A_N are fragments of the same length.

In general, an RMS value of no more than 2.5 Angstroms indicates identical structures (the breathing space of 2.5 Angstroms protects against small measurement errors which are quite common). Consequently, a small $v(P)$ value for a seqlet P indicates that the seqlet codes for identical local structures.

The approach described above was used on the set of seqlets obtained from the processing of the SwissProt Rel. 34 database. Out of the 565,432 seqlets in that set 342,820 seqlets were found to be matched by one or more sequences in PDB and 57,306 seqlets were matched by 2 or more PDB sequences. Almost 90% of this last group gets assigned an RMS error value of 2.5 Angstroms or less (Figure 5.6). Furthermore, several examples provide evidence for the structural specificity of seqlets. Figure 5.7 depicts such a case. The seqlet “V..G..G.G.T.L” is shown there along with the two sequences of PDB matching it. The two proteins are from different families. The specificity of the seqlet is deduced by the fact that the two backbones agree only over the amino acid regions that the seqlet covers while outside these regions the structures differentiate.

The process described above results in the generation of a library of 3D motifs, mapping short sequence descriptors (the seqlets) into local 3D structures. Such libraries can be used for the *prediction* of the structure of a new protein: given the sequence of the protein we identify all the seqlets from our library that are matched by the protein and then “patch” together the local structures that are associated with these seqlets in order to predict a global structure for the protein. We are currently in the process of experimenting with this approach. The preliminary results are promising.

The ability of the motif library to cover new proteins is of course directly dependent on its size. For this reason it is desirable to include as many 3D motifs as possible. A potential motif repository lies in the seqlets that are matched by exactly one PDB protein (there are $342,829 - 57,306 = 285,523$ of them in SwissProt Rel. 34). For each one of these seqlets a three dimensional structure is available from the PDB protein which matches that seqlet. What is not known is the extent to which we can conclude that *all* regions matching that seqlet will also have this structure. This requires a closer analysis of those seqlets which although found to be matched by two or more of the PDB proteins, they induced alignments with an RMS error of more than 2.5 Angstroms. We have, for example, observed that many of these patterns contain the amino acid Proline. This particular amino acid is known to be forcing “unnatural” turns into the structure of proteins because its side group is bound to the nitrogen atom of the amino group in the backbone part of the amino acid. Eventually, we would like to come up with a set of rules that allow us to decide with a high degree of certainty when a seqlet matched by only one sequence in PDB can become part of the motif library.

We also plan to keep on exploiting our sets of seqlets towards the annotation of uncharacterized proteins, along the lines we used for the annotation of *YZ28_METJA* in Section 4.5.2. Among the sequences that we looked at in an effort to validate the homology searching tool presented in Chapter 4 was the BRC2 protein (SwissProt id: *BRCA_HUMAN*), one of the two known breast cancer repressors. It is known that this protein is related to repairing DNA damaged by cancerous agents but it is not known exactly how it works. Using seqlets from the SwissProt database we proposed a model and presented it to researchers from the Memorial Sloan Katering hospital; they are currently carrying out experiments that will either validate or refute our findings.

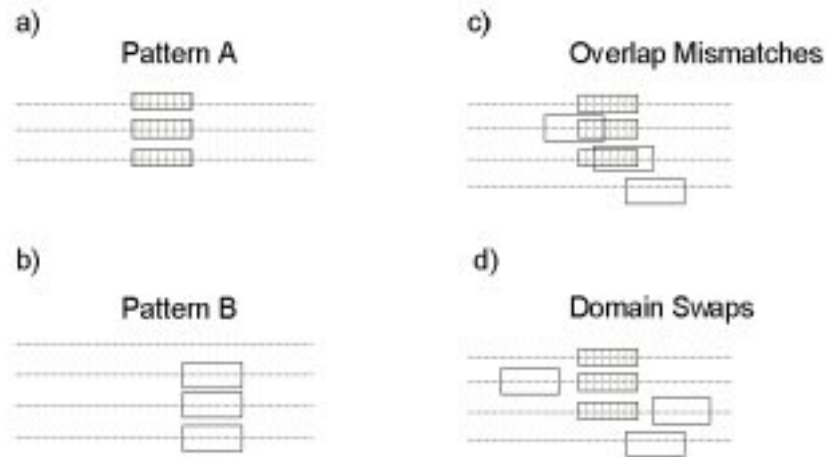


Figure 5.1: Two patterns A and B are shown on a set of input sequences. Every pattern by itself induces naturally an alignment of the sequences ((a) and (b)). Problems arise, however, when trying to synthesize an alignment that respects both patterns. The problems can be either because the intersections of the two patterns are not consistent (c) or because their relative order is not the same in all the sequences that match both the patterns (d).

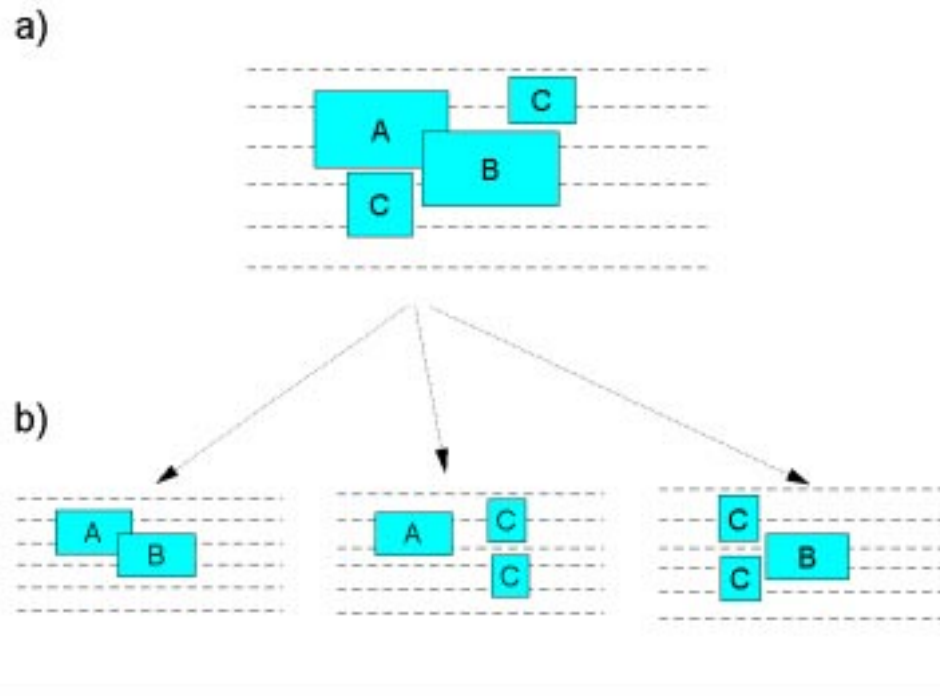


Figure 5.2: (a) Three patterns A, B, C shown relative to a set of input sequences. The ordering (A, C, B) is a consistent one; whenever a sequence matches more than one pattern the matching offsets respect the ordering. However, there is no feasible ordering of the set $\{A, B, C\}$. The only possible global alignments induced by these patterns are shown in (b). None of them respects all three patterns.

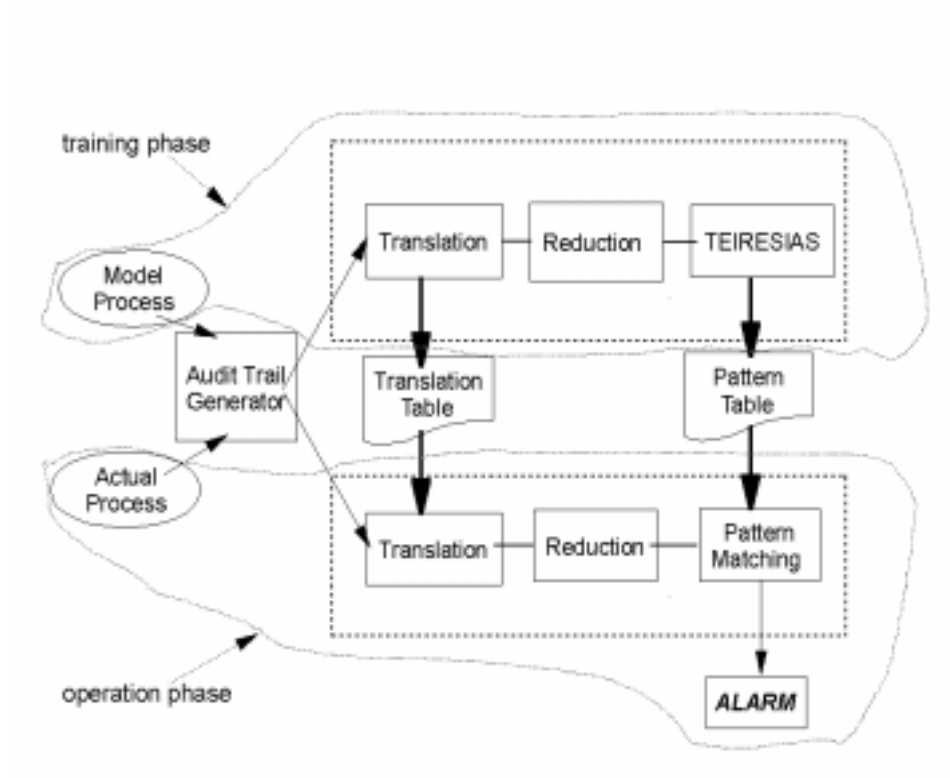


Figure 5.3: The architecture of the intrusion detection system.

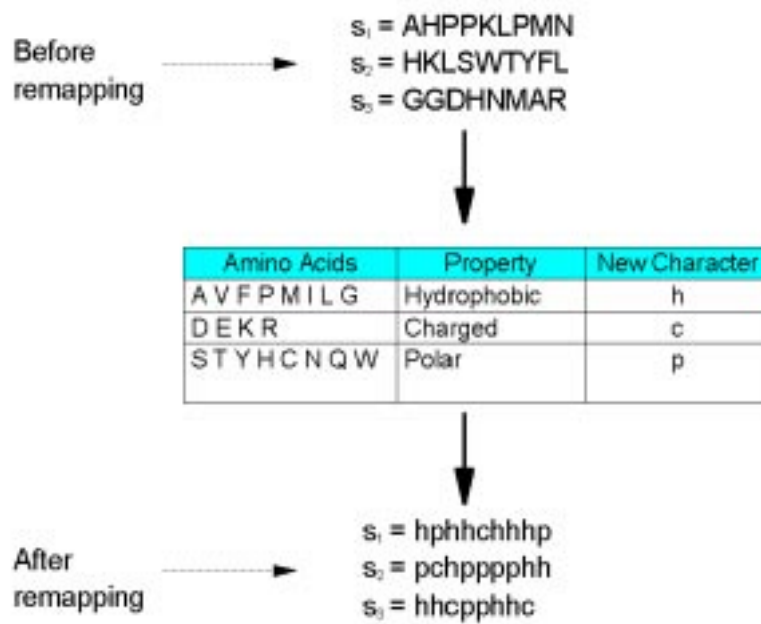


Figure 5.4: The classification of amino acids into disjoint groups of hydrophobic, charged and polar amino acids (see also Table 1.1) is providing a reduced alphabet for rewriting a set of three sequences.

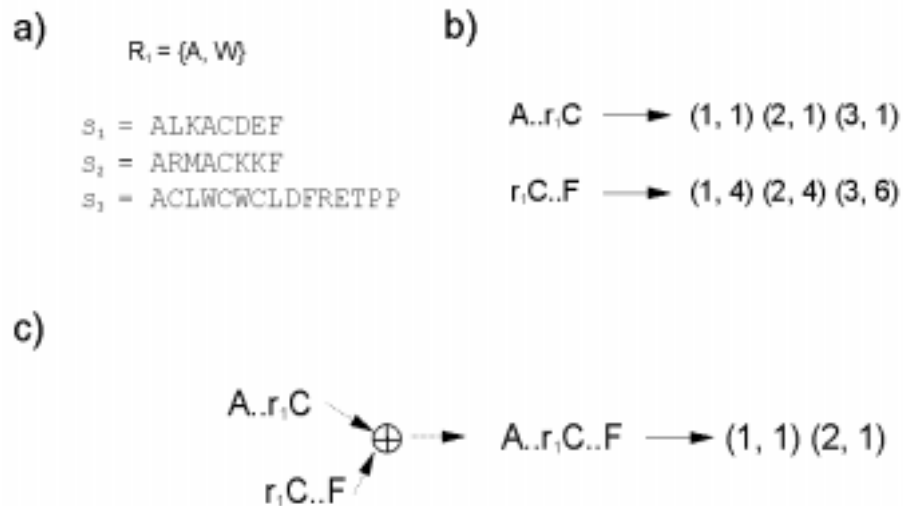


Figure 5.5: (a) An input consisting of the three sequences s_1, s_2, s_3 and only one group of related amino acids, namely the group $R_1 = \{A, W\}$. The character r_1 is used to denote the bracketed expression $[AW]$ introduced by R_1 .

(b) Two elementary patterns containing the new character r_1 along with their offset lists.

(c) The result of convolving the two above patterns is the new pattern “ $A.r_1C..F$ ”. All the input sequences, though, that match this pattern have only the character ‘A’ at the pattern position denoted by r_1 . In order to avoid subsequent unnecessary convolutions, r_1 should be replaced by the character A in the new pattern.

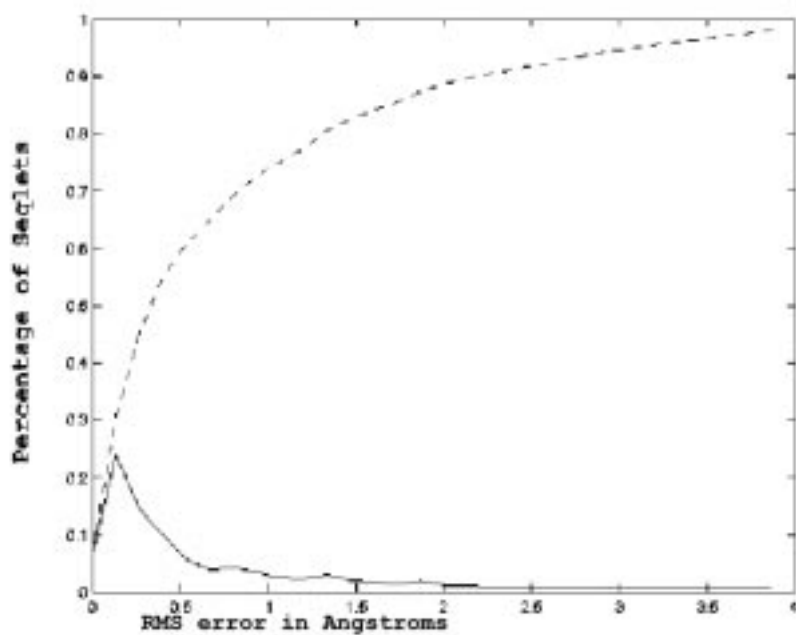


Figure 5.6: Distribution of RMS errors for the 57,306 seqlets from SwissProt that are matched by two or more sequences of the PDB database. The solid line describes the percentage of seqlets with a given RMS error while the broken line gives the cumulative RMS error.

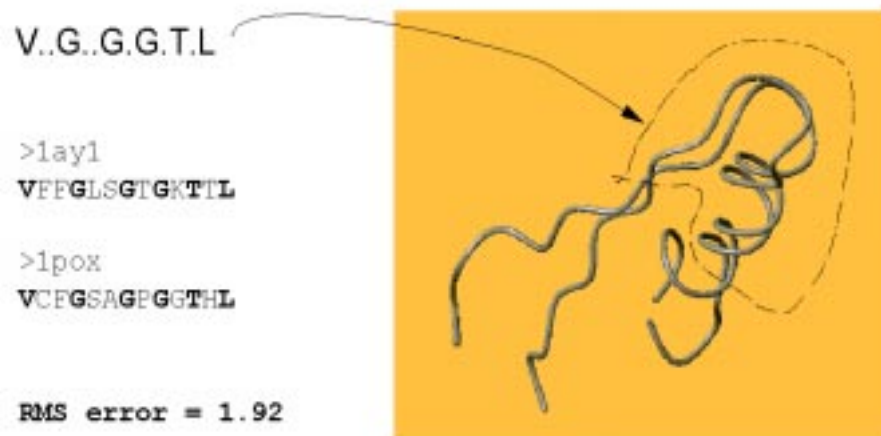


Figure 5.7: The seqlet “V..G..G.G.T.L” is matched by two PDP proteins that belong to different families: *1ayl* is a phosphoenolpyruvate kinase from the organism E.Coli while *1pox* is a pyruvate oxidase from Lactobacillus Plantarum. The backbone regions matching the seqlet were extracted from both proteins and they were aligned. The aligned regions (shown here encircled by the dotted line) have an RMS error of 1.92 Angstroms. It is interesting to notice that while the two backbones are very similar in the area delineated by the seqlet, they immediately become divergent outside of it.

Bibliography

- [1] S. Altschul. Gap costs for multiple sequence alignments. *Journal of Theoretical Biology*, 138:297–309, 1989.
- [2] S. Altschul, M. Boguski, W. Gish, and J.C. Wooton. Issues in searching molecular sequence databases. *Nature Genetics*, 6:119–129, 1992.
- [3] S. Altschul, W. Gish, W. Miller, E.W. Myers, and D. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215:403–410, 1990.
- [4] S. Altschul, T.L. Madden, A.A Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.
- [5] G. Arents and E.N. Moudrianakis. Topography of the histone octamer surface: repeating structural motifs utilized in the docking of nucleosomal dna. *Proceedings of the National Academy of Sciences*, 90:10489–10493, 1993.
- [6] T.K. Attwood and M.E. Beck. Prints - a protein motif fingerprint database. *Protein Engineering*, 7:841–848, 1994.
- [7] O.T. Avery, C.M. MacLeod, and M. McCarty. Studies on the chemical nature of the substance inducing transformation of pneumococcus types. *Journal of Experimental Medicine*, 79:137–158, 1944.
- [8] A. Bairoch and R. Apweiler. The swiss-prot protein sequence data bank and its supplement trembl in 1998. *Nucleic Acids Research*, 26:38–42, 1998.

- [9] A. Bairoch, P. Bucher, and K. Hofmann. The prosite database: its status in 1995. *Nucleic Acids Research*, 24:189–196, 1996.
- [10] A. Baxevanis and F. Ouellette, editors. *Bioinformatics: A Practical Guide to the Analysis of Genes and Proteins*. Wiley–Interscience, 1998.
- [11] A.D. Baxevanis, G. Arents, E.N. Moudrianakis, and D. Landsman. A variety of dna-binding and multimeric proteins contain the histone fold motif. *Nucleic Acids Research*, 23:2685–2691, 1995.
- [12] A.D. Baxevanis and D. Landsman. Histone and histone fold sequences and structures: a database. *Nucleic Acids Research*, 25:272–273, 1997.
- [13] F.R. Blattner, G. Plunkett III, C.A. Bloch, N.T. Perna, and V. Burland. The complete genome sequence of escherichia coli k-12. *Science*, 277:1453–1474, 1997.
- [14] P. Bork and T.J. Gibson. Applying motif and profile searches. *Methods in Enzymology*, 266:162–184, 1996.
- [15] P. Bork and M. Sudol. The ww domain: a signaling site in dystrophin? *Trends in Biochemistry*, 19:531–533, 1994.
- [16] E. Bornberg-Bauer, E. Rivals, and M. Vingron. Computational approaches to identify leucine zippers. *Nucleic Acids Research*, 26:2740–2746, 1998.
- [17] C. Braden and J. Tooze. *Introduction to Protein Structure*. Garland Publishing, 1991.
- [18] A. Brazma, I. Jonassen, I. Eidhammer, and D. Gilbert. Approaches to the automatic discovery of patterns in biosequences. Technical report, Department of Informatics, University of Bergen, Norway, 1995.
- [19] C.J. Bult, O. White, G.J. Olsen, L. Zhou, R.D. Fleischmann, G. Sutton, et al. Complete genome sequence of the methanogenic archaeon, methanococcus jannaschii. *Science*, 273:1058–1073, 1996.
- [20] C. Caskey, R. Eisenberg, E. Lander, and J. Straus. Hugo statement on patenting of dna. *Genome Digest*, 2:6–9, 1995.

- [21] C. Chothia. One thousand families for the molecular biologist. *Nature*, 357:543–544, 1992.
- [22] V. Chvatal. A greedy heuristic for the set covering problem. *Math. Oper. Res.*, 4:233–235, 1979.
- [23] F. Collins, A. Patrinos, E. Jordan, A. Chakravarti, R. Gesteland, and L. Walters. New goals for the us human genome project: 1998-2003. *Science*, 282(5389):682–689, 1998.
- [24] F. Corpet, J. Gouzy, and D. Kahn. The prodom database of protein families. *Nucleic Acids Research*, 26:323–326, 1998.
- [25] M.O. Dayhoff, R.M. Schwartz, and B.C. Orcutt. A model of evolutionary change in proteins. *Atlas of Protein Science and Structure*, 5:345–352, 1978.
- [26] H. Debar, M. Dacier, M. Nassehi, and A. Wespi. Fixed vs. variable-length patterns for detecting suspicious process behavior. In *ESORICS*, 1998.
- [27] A. Dembo and S. Karlin. Strong limit theorems of empirical functions for large exceedances of partial sums of iid variables. *The Annals of Probability*, 19:1737–1755, 1991.
- [28] R.F. Doolittle. What we have learned and will learn from sequence databases. In G. Bell and T. Marr, editors, *Computers and DNA*, pages 21–31. Addison–Wesley, 1990.
- [29] R.F. Doolittle. Convergent evolution: the need to be explicit. *Trends in Biochemistry*, 1:15–18, 1994.
- [30] R.F. Doolittle. The multiplicity of domains in proteins. *Annual Reviews in Biochemistry*, 64:287–314, 1995.
- [31] R.D. Fleischmann, M.D. Adams, O. White, R.A. Clayton, E.F. Kirkness, A.R. Kerlavage, et al. Whole-genome random sequencing and assembly of haemophilus influenzae rd. *Science*, 269:496–512, 1995.

- [32] S Forrest et al. A sense of self for unix processes. In *IEEE Symposium on Security and Privacy*, pages 120–128, 1996.
- [33] C.M. Fraser, J.D. Gocayne, O. White, M.D. Adams, et al. The minimal gene complement of mycoplasma genitalium. *Science*, 270:397–403, 1995.
- [34] M.R. Garey and D.S. Johnson. *Computers and Intractability: a Guide to the Theory of NP-Completeness*. W.H.Freeman, 1979.
- [35] D.G. George, W.C. Barker, and L.T. Hunt. Mutation data matrix and its uses. *Methods in Enzymology*, 183:338, 1990.
- [36] E.M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.
- [37] G.H. Gonnet, M.A. Cohen, and S.A. Benner. Exhaustive matching of the entire protein sequence database. *Science*, 256:1443–1445, 1992.
- [38] P. Green, D. Lipman, L. Hillier, R. Waterston, D. States, and J.M. Claverie. Ancient conserved regions in new gene sequences and the protein databases. *Science*, 259:1711–1716, 1993.
- [39] M. Gribskov, R. Luthy, and D. Eisenberg. Profile analysis. *Methods in Enzymology, Molecular Evolution: Computer Analysis of Protein and Nucleic Acid Sequences*, 183:146–159, 1990.
- [40] M. Gribskov, A. McLachlan, and D. Eisenberg. Profile analysis detection of distantly related proteins. *Proceedings of the National Academy of Sciences*, 88:4355–4358, 1987.
- [41] D. Gusfield. *Algorithms on Strings, Trees and Sequences: Computer Science and Computational Biology*. Cambridge University Press, 1997.
- [42] N.L. Harris, L. Hunter, and D.J. States. Megaclassification: Discovering motifs in massive datastreams. *10th National Conference on Artificial Intelligence*, pages 837–842, 1992.

- [43] S. Henikoff, E.A. Greene, S. Pietrokovski, P. Bork, T. K. Attwood, and Hood. L. Gene families: The taxonomy of protein paralogs and chimeras. *Science*, 278:609–614, 1997.
- [44] S. Henikoff and J. Henikoff. Automatic assembly of protein blocks for database searching. *Nucleic Acids Research*, 19:6565–6572, 1991.
- [45] S. Henikoff and J. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of the National Academy of Sciences*, 89:915–919, 1992.
- [46] S. Henikoff and J. Henikoff. Protein family classification based on searching a database of blocks. *Genomics*, 19:97–107, 1994.
- [47] A.D. Hershey and M. Chase. Independent function of viral protein and nucleic acid on growth of bacteriophage. *Journal of Genetic Physiology*, 36:39–56, 1952.
- [48] P. Hieter and M. Boguski. Functional genomics: It’s all how you read it. *Science*, 278(5338):601–602, 1997.
- [49] R. Himmelreich, H. Hilbert, H. Plagens, E. Pirkl, B.C. Li, and R. Herrmann. Complete sequence analysis of the genome of the bacterium mycoplasma pneumoniae. *Nucleic Acids Research*, 24:4420–4449, 1996.
- [50] Martinez H.M. An efficient method for finding repeats in molecular sequences. *Nucleic Acids Research*, 11:4629–4634, 1983.
- [51] Martinez H.M. A flexible multiple sequence alignment program. *Nucleic Acids Research*, 16:1683–1691, 1988.
- [52] T.C. Hodgman. The elucidation of protein function by sequence motif analysis. *Computer Applications in Biosciences*, 5:1–13, 1989.
- [53] L. Holm and C. Sander. Removing near-neighbor redundancy from large protein sequence collections. *Bioinformatics*, 14:423–429, 1998.
- [54] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation*. Addison–Wesley, 1979.

- [55] P. Horton and K. Nakai. Better prediction of protein cellular localization sites with the k nearest neighbors classifier. *International Conference on Intelligent Systems for Molecular Biology*, 5:147–152, 1997.
- [56] L.C.K. Hui. Color set size problem with applications to string matching. In *Combinatorial Pattern Matching*, pages 230–243, 1992.
- [57] I. Jonassen, J.F. Collins, and D.G. Higgins. Finding flexible patterns in unaligned protein sequences. *Protein Science*, 4:1587–1595, 1995.
- [58] T. Kaneko and S. Tabata. Complete genome structure of the unicellular cyanobacterium *synechocystis* sp. pcc6803. *Plant Cell Physiology*, 38:1171–1176, 1997.
- [59] O.H. Kapp, L. Moens, J. Vanfleteren, C.N. Trotman, T. Suzuki, and S.N. Vinogradov. Alignment of 700 globin sequences: Extent of amino acid substitution and its correlation with variation in volume. *Protein Science*, 4:2179–2190, 1995.
- [60] S. Karlin and S. Altschul. Methods for assessing the statistical significance of molecular sequence features by using general scoring schemes. *Proceedings of the National Academy of Sciences*, 87:2264–2268, 1990.
- [61] S. Karlin, A. Dembo, and T. Kawabata. Statistical composition of high-scoring segments from molecular sequences. *The Annals of Statistics*, 2:571–581, 1990.
- [62] A.P. Kosoresow and S.A. Hofmeyr. Intrusion detection via system call traces. *IEEE Software*, pages 35–42, 1997.
- [63] A. Krause and M. Vingron. A set-theoretic approach to database searching and clustering. *Bioinformatics*, 14:430–438, 1998.
- [64] A. Krogh, M. Brown, I. Mian, K. Sjolander, and D. Haussler. Hidden markov model in computational biology: Applications to protein modelling. *Journal of Molecular Biology*, 235:1501–1531, 1994.
- [65] J.B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–27, 1964.

- [66] J.B. Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrica*, 29:115–129, 1964.
- [67] I. Ladunga. Phylogenetic continuum indicates ‘galaxies’ in the protein universe: preliminary results on the natural group structures of proteins. *Journal of Molecular Evolution*, 34:358–375, 1992.
- [68] R. Lathrop, T. Webster, R. Smith, P. Winston, and T. Smith. Integrating ai with sequence analysis. In L. Hunter, editor, *Artificial Intelligence and Molecular Biology*, pages 211–258. AIII Press – MIT Press, 1993.
- [69] A.M. Lesk. Computational molecular biology. In A. Kent and J.G. Williams, editors, *Encyclopedia of Computer Science and Technology*, volume 31, pages 101–165. Marcel Dekker, 1994.
- [70] B. Lewin. *Genes VI*. Oxford University Press, 1997.
- [71] M. Linial, N. Linial, N. Tishby, and G. Yona. Global self–organization of all known protein sequences reveals inherent biological signatures. *Journal of Molecular Biology*, 268:539–556, 1997.
- [72] D. Lipman and W.R. Pearson. Rapid and sensitive protein similarity searches. *Science*, 227:1435–1441, 1985.
- [73] D. Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM*, 25:322–336, 1978.
- [74] Schmid M.B. More than just ‘histone-like’ proteins. *Cell*, 63:451–453, 1990.
- [75] A.Z. Murzin, S.E. Brenner, T. Hubbard, and C. Chotia. Scop: A structural classification of proteins data base for the investigation of sequences and structure. *Journal of Molecular Biology*, 247:536–540, 1995.
- [76] S.B. Needleman, , and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

- [77] A.F. Neuwald and P. Green. Detecting patterns in protein sequences. *Journal of Molecular Biology*, 239:698–712, 1994.
- [78] C.G. Nevill-Manning, T.W. Wu, and D.L. Brutlag. Highly specific protein sequence motifs for genome analysis. *Proceedings of the National Academy of Sciences*, 95:5865–5871, 1998.
- [79] C.G. Neville-Manning, K.S. Sethi, D. Wu, and D.L. Brutlag. Enumerating and ranking discrete motifs. In *International Conference on Intelligent Systems for Molecular Biology*, 1997.
- [80] J. Newmark. *Statistics and Probability in Modern Life*. Saunders College Publishing, 1997.
- [81] A. Ogiwara, I. Uchiyama, Y. Seto, and M. Kanehisa. Construction of a dictionary of sequence motifs that characterize groups of related proteins. *Protein Engineering*, 6:479–488, 1992.
- [82] C.A. Ouzounis and N.C. Kyrpides. Parallel origins of the nucleosome core and eukaryotic transcription from archaea. *Journal of Molecular Evolution*, 42:234–239, 1996.
- [83] W.R. Pearson. Protein sequence comparison and protein evolution. Tutorial of Intelligent Systems in Molecular Biology, 1995.
- [84] W.R. Pearson and D. Lipman. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences*, 85:2444–2448, 1988.
- [85] G.R. Reeck, E. Swanson, and D.C. Teller. The evolution of histones. *Journal of Molecular Evolution*, 10:309–317, 1983.
- [86] A. Robinson. *The story of writing*. Thames and Hudson, 1995.
- [87] M.A. Roytberg. A search for common patterns in many sequences. *CABIOS*, 8:57–64, 1992.

- [88] R. Rymon. Search through systematic set enumeration. *Third International Conference on Principles of Knowledge Representation and Reasoning*, pages 539–550, 1992.
- [89] M.F. Sagot and A. Viari. A double combinatorial approach to discovering patterns in biological sequences. *Proceedings of the 7th Symposium on Combinatorial Pattern Matching*, pages 186–208, 1996.
- [90] M.F. Sagot, A. Viari, and H. Soldano. Multiple sequence comparison: A peptide matching approach. *Proceedings of the 6th Symposium on Combinatorial Pattern Matching*, pages 366–385, 1995.
- [91] P. Salamon, J.C. Wooton, A.K. Konopka, and L. Hansen. On the robustness of maximum entropy relationships for complexity distributions of nucleotide sequences. *Computational Chemistry*, 17:135–148, 1993.
- [92] C. Sander and R. Schneider. Database of homology-derived protein structures and the structural meaning of sequence alignment. *Proteins*, 9:56–68, 1991.
- [93] M. Saraste, P.R. Sibbald, and A. Wittinghofer. The p-loop: a common motif in atp- and gtp-binding proteins. *Trends in Biochemistry*, 15:430–434, 1990.
- [94] S.R. Schmid and P. Linder. Dead protein family of putative rna helicases. *Molecular Microbiology*, 6:283–291, 1992.
- [95] H.O. Smith, T.M. Annau, and S. Chandrasegaran. Finding sequence motifs in groups of functionally related proteins. *Proceedings of the National Academy of Sciences*, 87:826–830, 1990.
- [96] R.F. Smith and T.F. Smith. Automatic generation of primary sequence patterns from sets of related protein sequences. *Nucleic Acids Research*, pages 118–122, 1990.
- [97] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
- [98] E. Sobel and M. Martinez. A multiple sequence alignment program. *Nucleic Acids Research*, 14:363–374, 1986.

- [99] R.J. Solomonoff. A formal theory of inductive inference, parts i and ii. *Information and Control*, 7:1–22, 224–254, 1964.
- [100] E.L. Sonnhammer, S.R. Eddy, and R. Durbin. Pfam: a comprehensive database of protein domain families based on seed alignments. *Proteins*, 28:405–420, 1997.
- [101] V.B. Strelets, I.N. Shindyalov, and H.A. Lim. Analysis of peptides from known proteins: clusterization in sequence space. *Journal of Molecular Evolution*, 39:625–630, 1994.
- [102] W.S. Sutton. The chromosome in heredity. *Biology Bulletin*, 4:231–251, 1903.
- [103] M. Suyama, T. Nishioka, and O. Jun'ichi. Searching for common sequence patterns among distantly related proteins. *Protein Engineering*, pages 1075–1080, 1995.
- [104] M. Suyama, T. Nishioka, and J. Oda. Searching for common sequence patterns among distantly related proteins. *Protein Engineering*, 8:1075–1080, 1995.
- [105] R.L. Tatusov, E.V. Koonin, and D.J. Lipman. A genomic perspective on protein families. *Science*, 278:631–637, 1997.
- [106] G. Thode, J.A. Garcia-Ranea, and J. Jimenez. Search for ancient patterns in protein sequences. *Journal of Molecular Evolution*, 42:224–233, 1996.
- [107] G. Vogel. Tracking the history of the genetic code. *Science*, pages 329–331, 1998.
- [108] J. Wang, T.G. Marr, D. Shasha, B.A. Shapiro, and G. Chirn. Discovering active motifs in sets of related protein sequences and using them for classification. *Nucleic Acids Research*, 22:2769–2775, 1994.
- [109] L. Wang and T. Jiang. On the complexity of multiple sequence alignment. *Journal of Computational Biology*, 1(4):337–348, 1994.
- [110] M.S. Waterman. Parametric and ensemble alignment algorithms. *Bulletin of Mathematical Biology*, 4:89–92, 1994.
- [111] M.S. Waterman. *Introduction to Computational Biology: Maps, Sequences, Genomes*. Chapman and Hall, 1995.

- [112] M.S. Waterman, D.J. Galas, and R. Arratia. Pattern recognition in several sequences: Consensus and alignment. *Bulletin of Mathematical Biology*, 46:515–527, 1984.
- [113] J.D. Watson and F. Crick. Molecular structure of nucleic acids: A structure for deoxyribose nucleic acid. *Nature*, 171:737–738, 1953.
- [114] J.D. Watson, N.H. Hopkins, J.W. Roberts, J. Steitz, and A.M. Weiner. *Molecular Biology of the Gene*. The Benjamin/Cummings Publishing Company, 1987.
- [115] N. Williams. Closing in on the complete yeast genome sequence. *Science*, 268:1560–1561, 1995.
- [116] J.C. Wooton and S. Federhen. Statistics of local complexity in amino acid sequences and sequence databases. *Computational Chemistry*, 17:149–163, 1993.
- [117] T.D. Wu and D.L. Brutlag. Identification of protein motifs using conserved amino acid properties and partitioning techniques. *International Conference on Intelligent Systems for Molecular Biology*, pages 402–410, 1995.
- [118] G. Yona, N. Linial, N. Tishby, and M. Linial. A map of the proteins space — an automatic hierarchical classification of all known proteins. In *International Conference on Intelligent Systems for Molecular Biology*, 1998.