University of Tunis
Higher Institute of Management

MASTER THESIS

Option: Computer science applied to management

Graph-Based Similarity of Protein Binding Sites Derived From Quasi-Clique Detection

Elaborated by:
Imen Boukhris

Advisors:
Zied Elouedi (ISG Tunis, Tunisia)
Eyke Hüllermeier (Philipps-Universität Marburg, Germany)

2008-2009

Acknowledgements

I would like to thank my advisor Zied Elouedi for introducing me to an interesting area of research. Thank you for your presence, your continuous guidance and for all the valuable discussions we have had during this research.

I am also thankful to my co-advisor Eyke Hüllermeier in the fulfillment of this master thesis for being very patient in supervising this work at all its stages. Your competence and your generosity have allowed me to work in ideal conditions. Your comments and your observations have always been of great interest. Thank you for your support and for your constant encouragements.

Special thanks go to Thomas Fober (member of the Knowledge Engineering and Bioinformatics Laboratory (KEBI) from Marburg University) for replying me each time I asked him questions regarding graph alignment. His e-mails have been very helpful. I am also thankful to all the other KEBI members, in particular to Florian Finkernagel, Marco Mernberger and Weiwei Cheng, for their hospitality and for the valuable discussions we have had about my research during my training in Germany.

Finally, I would like to express my sincere gratitude to all the professors of Tunis Higher Institute of business and to all the members of my laboratory LARODEC.

Contents

Introduction 1

I Theoretical aspect **5**

1 Basics of bioinformatics **7**
1.1 Introduction 7
1.2 Biological definitions 8
1.2.1 Deoxyribonucleic acid (DNA) molecule 8
1.2.2 Ribonucleic acid (RNA) molecule 8
1.2.3 Protein molecule 9
1.3 Uses of Bioinformatics 13
1.3.1 Fields 13
1.3.2 Bioinformatics applications 13
1.4 Bioinformatics spectrum 14
1.5 Research topics in Bioinformatics and Algorithms 16
1.6 Protein Alignment 21
1.6.1 Sequential Alignment 21
1.6.2 Structural Alignment 25
1.7 Conclusion 27

2 Graph matching and similarity **29**
2.1 Introduction 29
2.2 Graph matching problem 29
2.2.1 Graph isomorphism 30
2.2.2 Subgraph isomorphism 30
2.2.3 Maximum common subgraph 30
2.2.4 Minimum common supergraph 30
2.2.5 Graph edit distance 31
2.3 Complexity of graph matching 31
2.3.1 Complexity classes P, NP 31
2.3.2 Graph isomorphism 32
2.3.3 Subgraph isomorphism 32
2.3.4 Error-tolerant graph matching 33

2.4 Graph matching reformulation: Clique-detection . . . 33
2.4.1 Motivation . . . 33
2.4.2 Clique definition . . . 33
2.4.3 Clique detection . . . 33
2.5 An Error-tolerant graph matching method for the multiple structural alignment 34
2.5.1 Motivation . . . 34
2.5.2 Representation of graphs . . . 35
2.5.3 Construction of MGA . . . 35
2.5.4 Consensus graphs . . . 41
2.5.5 Classification . . . 42
2.6 Conclusion . . . 44

II Graph-similarity based on quasi-clique detection 45

3 Graph-similarity based on quasi-clique detection 47
3.1 Introduction . . . 47
3.2 Quasi-clique basics . . . 48
3.2.1 Notations . . . 48
3.2.2 Definitions . . . 48
3.3 Quasi-clique detection: QCD . . . 49
3.3.1 Difference with clique detection . . . 49
3.3.2 Quasi-clique detection . . . 50
3.3.3 Pruning techniques . . . 50
3.4 Graph similarity based on QCD . . . 51
3.4.1 Motivation . . . 51
3.4.2 Objective . . . 52
3.4.3 Product graph construction . . . 52
3.4.4 Similarity measure . . . 55
3.4.5 Graph-based similarity using QCD steps . . . 57
3.5 Conclusion . . . 65

4 Implementation and experimentation 67
4.1 Introduction . . . 67
4.2 Implementation . . . 67
4.2.1 Framework . . . 67
4.2.2 Principle variables . . . 68
4.2.3 Main programs . . . 70
4.2.4 Graph-based similarity derived from the quasi-clique detection algorithms 71
4.3 Experimentation . . . 76
4.3.1 Experimental Setup . . . 76
4.3.2 Evaluation criteria . . . 77
4.3.3 Validation criteria . . . 77
4.3.4 Simulation on real data sets . . . 77
4.4 Conclusion . . . 81
Conclusion . . . 83

A Alignment Algorithms **85**
A.1 Introduction 85
A.2 Sequence alignment 85
A.2.1 Pairwise sequence alignment algorithms 85
A.2.2 Multiple sequence alignment algorithms 91
A.3 Structural alignment 93
A.3.1 Pairwise structural alignment algorithms 93
A.3.2 Multiple structural alignment algorithms 94
A.4 Conclusion 95

B Cavbase database **97**
B.1 Introduction 97
B.2 CavBase 97
B.3 NADH-ATP dataset 98
B.3.1 Adenosine triphosphate- ATP 98
B.3.2 Nicotinamide adenine dinucleotide - NADH 99
B.4 Conclusion 99

References **101**

List of Figures

1.1 Levels protein structure 11
1.2 Interaction in binding site 12
1.3 Uses of Bioinformatics in Breadth and in Depth 15
1.4 Subtree from UPGMA: First step 16
1.5 Subtree from UPGMA: Second Step 17
1.6 A phylogenetic tree 17
1.7 Cartoon representation 18
1.8 Interaction network controlling the answer to the stress in carbon 20
1.9 The metabolic pathway for the biosynthesis of serine and glycine 20
1.10 DNA fragmentation and assembly 21
1.11 Alignment with Superposition 25

2.1 Clique detection for mcs problem 34
2.2 Pairwise graph alignment 38
2.3 Multiple graph alignment 39
2.4 Matrix representation of MGA and its corresponding graph alignment 40
2.5 Generation of an offspring solution 41
2.6 Fuzzy consensus graph 42
2.7 Classification 43

3.1 γ-quasi-clique 49
3.2 A quasi-clique and its subgraph 50
3.3 Set enumeration tree corresponding to the search space 50
3.4 Difference between clique and quasi-clique measure 52
3.5 Product graph nodes construction 53
3.6 triviality of γ-quasi-cliques 54
3.7 Product graph showing false-positive matches 55
3.8 Maximization of the common substructure size 55
3.9 Graph representing protein binding sites 58
3.10 Product graph 59
3.11 Connected components 60
3.12 The maximum common substructure between G1 and G2 65

4.1 Mean PCC according to φ values vs gamma values 79

4.2 Size maximum quasi-clique vs gamma value 80
4.3 Runtimes 80

A.1 Optimal path through dynamic programming matrix returns the multiple alignment 92

B.1 pseudocenters assignment 98
B.2 Adenosine triphosphate 99
B.3 Nicotinamide adenine dinucleotide 99

List of Tables

1.1 Amino acids notation 10
1.2 Assignments between amino acids and codons 10

3.1 G1 adjacency matrix 58
3.2 G2 adjacency matrix 58
3.3 Edge matrix 59

4.1 Description of the database 78
4.2 The impact of the variation ϕ values on the PCC 79

A.1 Dot matrix 86
A.2 F-Matrix initialization 86
A.3 F-Matrix 87
A.4 Optimal alignment on an F-Matrix 87
A.5 Smith-Waterman F-Matrix 88
A.6 FASTA: comparison between amino-acids positions 90

Introduction

The discoveries in biomedical and pharmaceutical researches have caused a fundamental progress in many fields like health care, nutrition and hygiene. Since biology knows a surge in molecular data, there was a need for computational methods to make biological investigations. For instance, they help to make simulations of molecular processes, to predict drugs effects and also to develop in silico models that will complement in vitro and in vivo biological experiments. Hence, an interdisciplinary area combining biology, computer science, mathematics and statics called bioinformatics appears to deal with the comprehension, the organization and the analysis of the information associated with biological molecules as (DNA[1]/RNA[2]) and proteins, on a large scale.

Many algorithms in bioinformatics focus on the protein analysis. This interest is related to the fact that proteins regulate all biological processes in an organism. They are the most abundant and diversified molecules present in cells. The human body has about one hundred thousand different proteins. Their importance is related with the diversity of their functions (oxygen transport, catalysis, immune system response, etc.) and with the fact that their biosynthesis depends directly on the genetic code. Hence, we can say that they are informative structures.

Proteins can be represented in a linear way (a sequence of amino-acids) or in the form of a graph that illustrates the protein structure. In binding sites cavities, proteins interact with binding partners which are complementary in shape and physicochemical properties. Graphs are frequently used to describe the geometry and the physicochemical composition of these protein active sites.

Proteins are classified into families according to their functions, however two proteins belonging to the same family cannot be exactly identical. For this reason, we expect similarities between them. Two binding sites are regarded as similar, if they share a common spatial arrangement and expose similar physicochemical properties into the binding site. One way to predict the function of a new protein is to analyze its structure and compare it with known stored proteins in one corresponding database (e.g. PDB[3], CavBase), measure its distance to the known proteins functional class and thus predict the family to which it belongs.

[1]DNA: Deoxyribonucleic acid
[2]RNA: Ribonucleic acid
[3]PDB: Protein Data Bank

One research axis in bioinformatics is alignment. An alignment is the process of lining up biomolecules in a best manner with which a maximal level of identity is achieved. Hence, region of similarities between them have to be identified. This process can be done for a pair of biomolecules and it is called pairwise alignment or for more than two: multiple alignment. One way to make a multiple alignment is to merge multiple pairwise alignments according to a pivot. These alignments can be made according to the level of representation of the biomolecules: sequence alignment and structural (or graph) alignment. Compared to sequence alignments, graph alignments present a much greater algorithmic challenge, because the vertices in a graph do not have a natural order which is the case with sequences where a sequence has a certain order.

Since similarity regions among biomolecule structures are sought, their corresponding graphs have to be compared. Graph matching techniques seem to be appropriate for this purpose. They allow the identification of the largest common substructure (the maximum common subgraph (mcs)) between graphs. One way to implement the mcs problem is to reformulate it onto another graph theory problem: clique-detection on a new structure called product graph where the maximum clique represents the maximum common substructure.

In this master thesis, we are looking for similarities among protein binding site structures, i.e. a common substructure between their corresponding graphs. In the clique-detection algorithm used for this purpose, the clique concept can be seen as too restrictive because of the noise in data sets, the distortions that may occur because of the flexible nature of the protein 3D-structure and mutations. In fact, cliques have the requirement that all nodes must be connected to all each others and thus it will limit the size of the common substructure.

Our aim through this work, is to make the structural comparison of the protein binding sites more efficient and optimize the common substructure size. We will use the quasi-clique concept, instead of clique, to measure the similarity between proteins independently of sequence or fold information. Quasi-cliques are a relaxation of cliques. A node is connected to a fixed number of nodes and not necessarily to all of them, as it was the case for cliques. This relaxation will help to obtain larger common substructures and thus produce a better quality alignment and improve classification results.

This report is organized in four chapters belonging to two main parts:

Part I: Theoretical aspects presents the necessary theoretical aspects concerning the basics of bioinformatics with a special focus on the protein alignment field in chapter 1, whereas chapter 2 details the graph-based similarity concept and the types of graph matching.

Part II: Details a graph-based similarity using quasi-clique detection with a similarity measure between two graphs. Chapter 3, shows how quasi-cliques measure efficiently the similarity between two structures. Chapter 4 is related to the implementation and simulations which have been performed in order to analyze and evaluate results given by our proposed method.

Finally, a conclusion summarizes all the works presented in this report and proposes fur-

ther works that may be done to improve our methods.

Two appendices are provided at the end of this report. The first one presents an outline of the algorithms used in literature to make alignments. The second one, presents description of the data used in simulations.

Part I

Theoretical aspect

Chapter 1

Basics of bioinformatics

1.1 Introduction

The progress in medicine and in drug design is related to discoveries in the biological science and also in computer science. In fact, information from the sequencing of genome, or from large databases about plants and animals can lead to discoveries and developments of new drugs. It is necessary to well comprehend and analyze biological data to obtain efficient and meaningful results.
Roughly speaking, quasi-cliques are "almost complete" graphs $G = (V, E)$. In the literature, different definitions of quasi-cliques have been proposed. Some of them are based on the degree of the nodes (Matsuda, Ishihara, & Hashimoto, 1999; Pei, Jiang, & Zhang, 2005), calling G a quasi-clique if every node in V is adjacent to at least $\gamma \cdot (|V| - 1)$ other nodes, where $deg(v)$ is the number of nodes adjacent to v.

$\gamma \cdot$ (|V| -1) Biology has known a surge in molecular data, especially since the sequencing of human genome. Indeed, about 50 terabyte of descriptive data relating to it have already been acquired. However, their conversion into operational information is a fundamental challenge (Royal Academy of Engineering, 2007). Bioinformatics appears from a need of the application of computational techniques to biological research to well organize and interpret the information associated to these macromolecules (Luscombe et al., 2001).

Even if, in the 1960's, Dayhoff and her collaborators at the National Biomedical Research Foundation were the first who assembled databases of protein's sequences organized in families and superfamilies based on their degree of similarity (Mount, 2004). The term bioinformatics was originally coined in the mid-1980 (Attwood & Parry-Smith, 1999).

Bioinformatics is composed of 'bio' which refers to biology and means the study of life and living organisms; and of 'informatics' which is the science of information, it is also referring to the application of computers to store and process information (Dreyfus, 1962) and to techniques like applied mathematics, computer science and statistics. Therefore, bioinformatics in not only the use of computers to handle biological information, but it has also to manage, process and understand the large amounts of data, information, and knowledge in biology.

The National Institutes of Health (NIH) Biomedical Information Science and Technology Initiative Consortium agreed on the following definition of bioinformatics as the research, development, or application of computational tools and approaches for expanding the use of biological, medical, behavioral or health data, including those to acquire, store, organize, archive, analyze, or visualize such data. So, in addition to studying biology in vivo, i.e., making real experience with the organism; in vitro (in the test tube), now it can also be done in silico (using computers).

This chapter is organized as follows: Section 1.2 presents a biological background to the field. Section 1.3 tells about the uses of bioinformatics. Types of analysis are discussed in Section 1.4. Section 1.5 introduces to some research topics and algorithms used in the bioinformatics area. Finally, Section 1.6 details a topic in bioinformatics called proteins alignment.

1.2 Biological definitions

Bioinformatics stores, organizes and develops tools, and resources that aid to the analysis of genes (DNA[1]/RNA[2]) and proteins (amino acids) and a subsequent interpretation of the results in a biologically significant way. In the following part, we present definitions of these data and the interactions between them.

1.2.1 Deoxyribonucleic acid (DNA) molecule

DNA molecules are composed of two polymers of nucleotides. Each polymer is the result of the attachment between the phosphate-deoxyribose backbone and one of these four bases: Adenine (A), Cytosine (C), Guanine (G) and Thymine (T).

The sequence of these four bases along the backbone encodes genetic information. Thus, DNA contains the instructions needed to construct other components such as proteins and RNA molecules. The DNA segments that carry this genetic information are called genes, they serve to protein's synthesis. Some of the genetic information is stored as the genetic code which is a set of three letters called 'codons' formed from a sequence of three nucleotides (e.g. ATC GTC AGA ATA) where each three-nucleotide combination stands for an amino acid (Crick, 1968).

Within cells, DNA is organized into structures called chromosomes. The set of genes within an organism is called genome. The study of genes expression and organism's entire genome is called genomics.

1.2.2 Ribonucleic acid (RNA) molecule

RNA and DNA are very similar, however while DNA is usually double-stranded, RNA is single-stranded and much shorter. RNA nucleotides contain ribose while DNA contains deoxyribose, which make RNA less stable. Besides, RNA has the base uracil (U) and no more

[1]DNA: Deoxyribonucleic acid
[2]RNA: Ribonucleic acid

thymine (T). Within a gene, the sequence of bases along a DNA strand is transcribed into messenger RNA (mRNA). We have to distinguish between the three types of RNA:

- messenger RNA (mRNA) is made by copying the gene from the DNA.
- transfer RNA (tRNA) is located in the cytoplasm. Its function lies mainly in transferring the mRNA.
- ribosomal (rRNA) is a part of the structure of a ribosome and it is the site where the protein is synthesized.

1.2.3 Protein molecule

As noticed by the hypothesis of *Central Dogma*, the cell can translate in only one direction: from nucleic acid to protein (Crick, 1968). A protein is an arrangement of amino acids joined by peptide bonds. Each amino acid has a physicochemical property, e.g. hydrogen-bond: Donor (DO), Acceptor (AC), Mixed donor/acceptor (DA), Hydrophobic aliphatic (AL) and aromatic (PI) (Schmitt et al., 2002).

The genetic material is first translated into RNA(mRNA), subsequently mRNA is translated into protein, so the relationship between the nucleotide sequences of genes and the amino-acid sequences of proteins is determined by the genetic code. Table 1.1 presents the notation of the 20 different amino acids and their one-letter representation and Table 1.2 illustrates how the genetic code specifies them (Crick, 1968).

Table 1.1: Amino acids notation

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

Table 1.2: Assignments between amino acids and codons

1st nucleotide → 2nd nucleotide ↓	U	C	A	G	3^{rd}nucleotide ↓
U	Phe	Ser	Tyr	Cys	U
	Phe	Ser	Tyr	Cys	C
	Leu	Ser	ochre		A
	Leu	Ser	ambre	Trp	G
C	Leu	Pro	His	Arg	U
	Leu	Pro	His	Arg	C
	Leu	Pro	GluN	Arg	A
	Leu	Pro	GluN	Arg	G
A	Ile	Thr	Asn	Ser	U
	Ile	Thr	Asn	Ser	C
	Ile	Thr	Lys	Arg	A
	Met	Thr	Lys	Arg	G
G	Val	Ala	Asp	Gly	U
	Val	Ala	Asp	Gly	C
	Val	Ala	Glu	Gly	A
	Val	Ala	Glu	Gly	G

Proteins play an important role within the organism. They are responsible of the most chemical reaction performed in the cell. For instance, sending messages to brain, building teeth and bone, making the heart beat or digesting food. They have many functions referring to the family they belong to it. It can be catalyzers as for enzymes, transporters as for hemoglobin, or those implied in cell adhesion as for collagen, etc. Even if the protein structure is more complex, it is more informational. We call proteomics the survey of the protein content of cells. Proteins are observed in four structures. Figure 1.1 shows the relation between them (Golan et al., 2008).

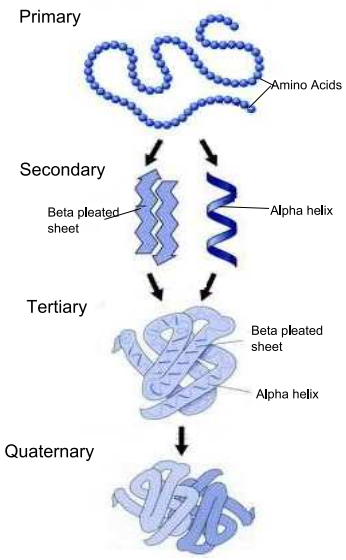


Figure 1.1: Levels protein structure

- **Primary structure:** it is a sequence of amino acids arranged in a linear way.
- **Secondary structure:**it is a protein is represented by regular local structures (alpha-helix, beta-sheets).
- **Tertiary structure:** it is a packing of secondary structures; a three-dimensional representation due to protein folding resulting from the interaction between distant amino acids. Thus, it is a spatial organization defined by atomic coordinates. 3D structures contain much more information compared to sequence. As shown in Figure **??**(A) (Alberts, 1998), the folding process creates cavities in the surface of a protein. Some of these cavities are defined as binding sites in which proteins interact with other binding partners(e.g. ligands) that are complementary in shape and physicochemical properties. Figure **??**(B) shows the interaction with the cyclic AMP ligand. We can identify the behavior of a cell and the function of a protein by its 3D structure, that is why we can affirm that the structure and the shape of a binding pocket are crucial to understand molecular interaction and thus protein's function. To determine protein's structure, X-ray crystallography and nuclear magnetic resonance are used however these techniques are time consuming and costly.

- **Quaternary structure:** it represents the interaction between polymers of proteins.

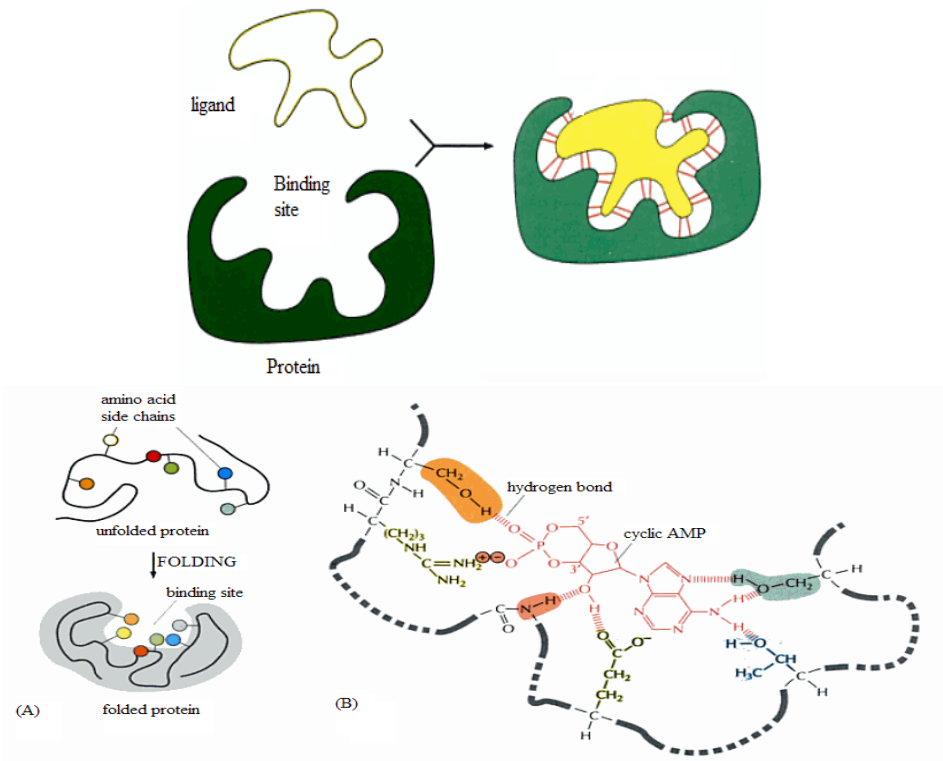


Figure 1.2: Interaction in binding site

1.3 Uses of Bioinformatics

Bioinformatics can be used to solve many biological problems, to find out how biological objects work and also to study evolutionary relationships between them.

1.3.1 Fields

Bioinformatics may be using in many fields as human health, agriculture, environment and biotechnology to advance research and development. It can offer benefits in the areas of molecular medicine to produce better and more personalized drugs. These drugs are customized to prevent or cure diseases. Bioinformatics will cause enormous changes in health care. In fact, treatments might be tailored to the patient and disease (Ohlstein et al., 2000). It can also be used for environmental purposes by identifying waste cleanup bacteria as well as alternative energy resources or studying climate changes. In agriculture it can be used for producing high yield low maintenance crops, insect resistance or to improve nutritional quality.

1.3.2 Bioinformatics applications

To achieve the goals mentioned above, we have to understand how they are related to the subdisciplines of bioinformatics. According to The National Center for Biotechnology Information (NBCI) there are three important sub-disciplines within bioinformatics. One subdiscipline concerns the development of new algorithms and statistics that enable to assess relationships among members of large data sets. The second subdiscipline lets the analysis and interpretation of various types of data including nucleotide and amino acid sequences, protein domains, and protein structures. The third subdiscipline allows the development and implementation of tools that enable efficient access and management of different types of information.

Thus, bioinformatics has to make a large-scale examination by transforming information from databases into a readable and understandable form, e.g. the recognition of the relation between structure folds and biochemical functions, it has also to develop algorithms and tools to analyze and interpret data. Here, we present some examples of fields and explain how bioinformatics treats them.

1.3.2.1 Evolutionary studies (phylogenies)

It means the study of the evolution of genes and genomes of organisms and populations and it can lead to identify their techniques of adaptation. Bioinformatics may serve to find homolog and therefor deduce protein's function: homologous are biomolecules with a common ancestor. They are related and therefore they share features. Biological principles of evolution are a great solution to deal with incomplete and noisy data and filter out useful information (Cohen, 2004). So, in order to detect homologous, similarities should be identified. Then, information can be transferred between them for example to poorly characterized or complicated ones. This is the case for structural data where theoretical models of proteins are usually based on experimentally solved structures of close homologous.

1.3.2.2 Pharmaceutical field and rational drug design

From homologous of amino acid sequences, we can predict the structure of a given protein based on known similar structures and then design a molecule that can bind this model structure. Finding discriminative features among proteins can help for drug design. It is the case for trypsin and thrombin which are closely related proteins of the family of serine proteases (Branden & Tooze, 1999). Trypsin is involved in the digestive system while thrombin is a coagulation protein. In order to identify the inhibitors of thrombin, it is important to detect difference between trypsin and thrombin.

1.3.2.3 Environmental field

Genomics data allow the identification of an environmental mode of stressor, the comprehension of how an organism responds to these stressors and the way to guide the selection of bioindicators for monitoring the impact of stressors on human and ecological health. Thus, genomics research tools will for example assist in predictive toxicology and will have an impact to assess the risk from exposure to stressors and improve our risk assessment (Environmental Protection Agency, 2002).

1.4 Bioinformatics spectrum

The bioinformatics spectrum ranges from collection and organization of biological databases to complex solutions. It means that from a gene sequence, we can determine the sequence and then the structure of a protein. With geometry calculations, we can further find out the protein's surface. Ligands that may bind the protein on its surface can be predicted using molecular simulation and structure docking. Hence, as we said before for the application of bioinformatics for pharmaceutical purposes for example, it is possible to design a specific drug.

Studying molecules can be done by examining one molecule at a time, making comparisons between a pairs of sequence of genes or proteins to identify evolutionary relationships or predict the structure. Comparisons can also be done between the structure of a pair of proteins. This may help to identify its function.

With a pair of proteins, we can make comparisons between the sequences and structures of evolutionary related proteins. With more data, we can use algorithms to create phylogenetic trees, which are trees that trace the evolution of the proteins for example. It is also possible to characterize a functional family and possibly detect discriminative features among these sequences or structures. Rather than detecting relationships between some molecules, information collected into databases can be used to trace evolutionary relationships of whole organisms. Figure 1.3 shows these types as an expansion of biological research in breadth and depth (Luscombe et al., 2001).

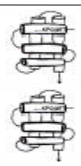
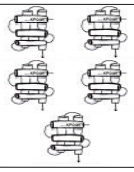
Depth: Rational Drug Design (Physics)		Breadth : Homologs, Large-scale Surveys, Informatics			
			Pairwise comparison, Sequence & Structure Alignment	Multiple alignments, patterns, templates, trees	Databases, scoring schemes, census
		1	2	3-100	100+
	Genome Sequence	actgcattggcatc	actgcattggcatc actgcattggcatc	actgcattggcatc actgcattgacatc aattcattggcatc acttgcatggcata	actgcattggcatc actgcattggcatc actgcattgacatc aattcattggcatc acttgcatggcata actgcattggcatc actgcattgacatc aattcattggcatc acttgcatggcata actgcattgacatc aattcattggcatc acttgcatggcata actgcattggcatc
	Gene Finding				
	Protein Sequence	ALMNAKKKPQ	ALMNAKKKPQ ALMNAKPKPQ	ALMNAKKKPQ ALMNAKPKPQ ALMMNAPKPQ ALMNAKKKKQ	ALMNAKKKPQ ALMNAKPKPQ ALMMNAPKPQ ALMNAKKKKQ ALMNAKKKPQ ALMNAKPKPQ ALMMNAPKPQ ALMNAKKKKQ
	Structure prediction				
	Protein Structure				
	Geometry calculation				
	Protein Surface				
	Molecular simulation				
	Force Field				
	Structure docking				
	Ligand Complex				

Figure 1.3: Uses of Bioinformatics in Breadth and in Depth

1.5 Research topics in Bioinformatics and Algorithms

Implementing algorithms and tools will allow detection of relations between molecules on large-scale. As relations between molecules, we can mention structural and functional similarities which will help to define patterns and thus characterize families, evolutionary relationships or interactions between them within an organism. Among these algorithms that are developed in bioinformatics and cited in (Cohen, 2004), there are those that focus on:

- **Constructing evolutionary tree**. These phylogenetic trees are built to establish evolution and thus homologies among a family of related nucleic acid or protein sequences. They have to group sequences according to their degree of similarity. In a phylogenetic tree, each node with descendants represents the most recent common ancestor of the descendants, and the edge lengths in some trees correspond to time estimates. As algorithms that can be used for this purpose, we can mention agglomerative algorithms as UPGMA (Unweighted Pair Group Method with Arithmetic averages) (Sneath & Sokal, 1978) or neighbor-joining algorithm (Saitou & Nei, 1987). UPGMA is a simple agglomerative or bottom-up data clustering method. The algorithm examines the structure present in a pairwise distance matrix to then construct a rooted tree (dendrogram). At each step, the nearest two elements (clusters), i.e. having lowest distances, are combined into a higher-level cluster. A new matrix distance is constructed in which the distance between any two clusters A and B is taken to be the average of all distances between pairs of objects "a" in A and "b" in B.

  Example 1.1. *Given six sequences denoted A, B, C, D, E, F. The following matrix represents the distance between each distinct pairs of these sequences.*

$$\left|\begin{array}{cccccc} & A & B & C & D & E \\ B & 4 & & & & \\ C & 8 & 8 & & & \\ D & 10 & 10 & 10 & & \\ E & 20 & 20 & 20 & 10 & \\ F & 20 & 20 & 20 & 20 & 20 \end{array}\right|$$

  This distance matrix, will be used to construct a phylogenetic tree using the UPGMA method. The distance between two composites X and Y will be notated as dist(X,Y).

  At the beginning, we cluster the pair of sequence with the smallest distance, here it is A and B that are separated by a distance of 4. The branching point is positioned at a distance of 4 / 2 = 2 substitution. We thus construct a subtree as depicted in Figure1.4:

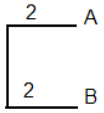


Figure 1.4: Subtree from UPGMA: First step

The first clustering A and B are considered as a single composite (A,B) and now, we calculate the new distance matrix as follows:

dist((A,B),C) = (dist(A,C) + dist(B,C)) / 2 = 8
dist((A,B),D) = (dist(A,D) + dist(B,D)) / 2 = 10
dist((A,B),E) = (dist(A,E) + dist(B,E)) / 2 = 20
dist((A,B),F) = (dist(A,F) + dist(B,F)) / 2 = 20

$$\left|\begin{array}{ccccc} & A,B & C & D & E \\ C & 8 & & & \\ D & 10 & 10 & & \\ E & 20 & 20 & 10 & \\ F & 20 & 20 & 20 & 20 \end{array}\right|$$

This matrix will be used to extend the subtree as shown in Figure 1.5:

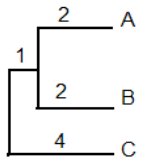


Figure 1.5: Subtree from UPGMA: Second Step

Pushed by the idea of a common ancestor, we will execute the algorithm until all the nodes are collapsed into a single node. Figure 1.6 shows the resultant evolutionary tree:

$$\left|\begin{array}{cc} & ABC,DE \\ F & 20 \end{array}\right|$$

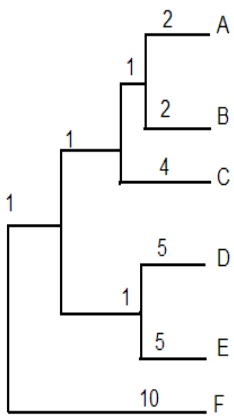


Figure 1.6: A phylogenetic tree

- **Detecting patterns in sequences**. It means delimiting parts of sequences that have a biological meaning (e.g. detecting genes in DNA or alpha-sheets in protein) (Cohen, 2004).

  This can be done with grammars and parsing or with Hidden Markov Models. As presented in (Searls, 1992), linguistic is related to biology. In fact, it can be applied to biology to represent the nondeterminism present in biological sequences. Grammars are used as models for biological sequences. They are defined by a set of productions that are used to make rules to make description. Parsing is a technique of retracing the generation of a sentence using a given grammar rules, it may produce a structure description ("parse tree") for correct sentences.

  Example 1.2. *To describe syntax we often use Context-Free Grammars (CFG). Here we present an example of CFG defining a protein sequence which is a linear combination of letters representing amino acids and gaps:*
  $Start \rightarrow Gap\ R\ V\ V\ V\ D\ V\ Gap,$
  $Gap \rightarrow \epsilon \mid X\ Gap,$
  $X \rightarrow A\ |R|N|D|C|...|V$

- **Predicting 3D structure**. 3D structures are important in biology. They allow us to understand the interaction and thus the function of molecules. However, determining such a structure by X-ray crystallography for example is very expensive and time consuming. Bioinformatics proposed many methods that have served to find out 3D structures. Among them, we can mention cartoons (Gilbert et al., 1999) which are representations for proteins. Cartoons illustrate the topology of proteins through a graph where sheets are represented by triangles, helices by circles and lines joining them express their 3D proximity.

  Example 1.3. *In this example, we present the cartoon representation of the 2bop protein (see Figure 1.7).*

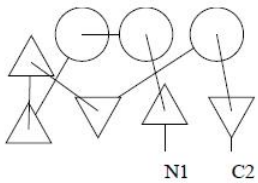


Figure 1.7: Cartoon representation

- **Inferring cell regulation**. There are an importance to understand the functions and interactions between cells and therefore genes. Many algorithms in bioinformatics were developed for this purpose. Simulation and modeling of gene interaction for example, can solve this problem. In fact, this approach simulates genes' interactions and compare results with experimental data to verify that the model used in simulation is correct. In modeling, procedures for the interaction process have to be described.

  Example 1.4. *The network considered in this example, is the one controlling to the stress in carbon at the Escherichia Coli (see Figure 1.8). In the simulation step, the*

genes, proteins involved are represented in an interacting network. The modeling step, a linear differential equation was used to describe the relationship between the rrn operon and FIS protein.

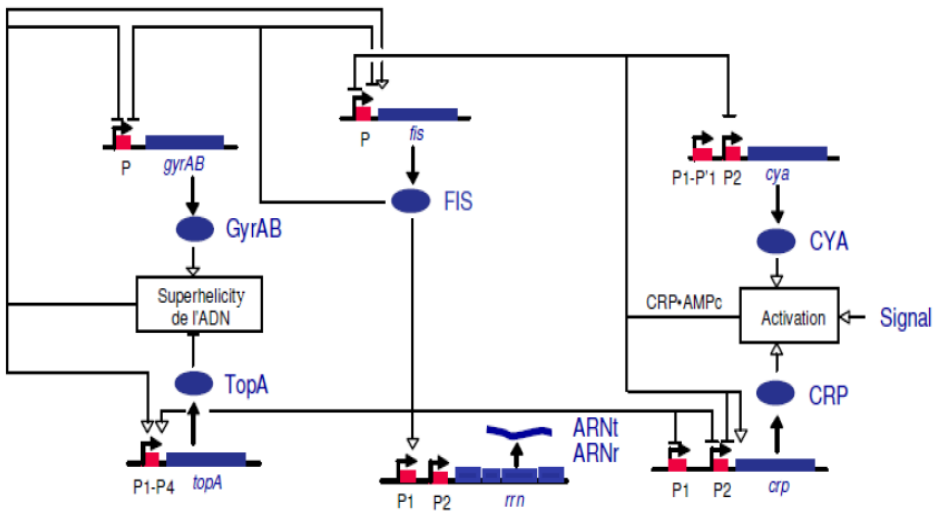


Figure 1.8: Interaction network controlling the answer to the stress in carbon

- **Determining protein function and metabolic pathways**. A function requires the cooperative action of several molecules. Thus, the protein function is related to its role in metabolic pathways which are series of linked chemical reactions occurring within a cell (an interactive system), e.g. production of ATP molecule in the Mitochondria. The reactants of one reaction are the products of the previous one. These pathways can be implemented in the form of databases, e.g EcoCyc database for the bacterium Escherichia coli K-12 MG1655 and can be modeled through graphs, where nodes represent chemical reactions, proteins and enzymes that are involved (Karp, 2001). Thus, these graphs ease answering queries like determining reaction where a given enzyme acts.

  Example 1.5. *We present in this example the metabolic pathway for the biosynthesis of two amino acids (serine and glycine) for the E.Coli K12 bacterium. In Figure1.9, the different transformations are presented.*

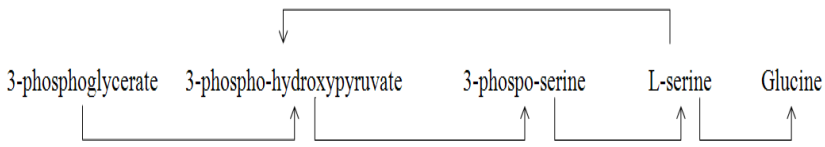


Figure 1.9: The metabolic pathway for the biosynthesis of serine and glycine

- **Assembling DNA fragments**. It means creating tools to combine partial information of DNA (fragments) into a consistent map. Genetic algorithms have been explored to the problem of DNA fragments assembly (Parsons et al., 1995).

  Example 1.6. *Since DNA chromosomes are too long, it is necessary to decomposition them into fragments to analyze them. These fragments, on which studies are made, are randomly generated. In order to assemble these fragments again, the end of these sequences should be determined (see Figure 1.10). Once, the end-sequences are identified, it will be easy to assemble fragments into a continuous sequence of larger size.*

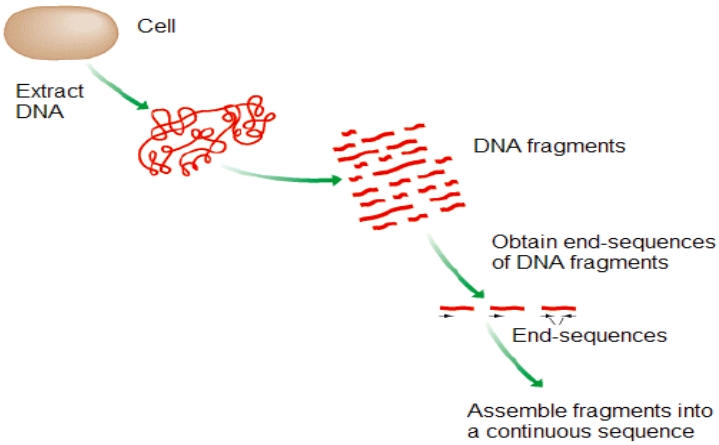


Figure 1.10: DNA fragmentation and assembly

- **Comparing biomolecules and making an alignment among them** It can lead to discovery of evolution relationships, structural or even functional predictions. Alignment methods are used to compare biomolecules and to express their degree of similarity. An alignment is the process of lining them up to achieve a maximal level of identity. This is done by the identification of similarities' regions that may be a consequence of functional relationships. Alignment can be calculated for only pairs of molecules which is called pairwise alignment or for more than two and it is called multiple alignments. This topic will be detailed in the next section.

1.6 Protein Alignment

Even if proteins are more complex than DNA or RNA and due to their importance, many algorithms in bioinformatics focus on their analysis. They regulate all biological processes in a cellular organism (Branden & Tooze, 1999). The comparison of proteins, either in sequence or structure, is the most fundamental technique in computational biology (Koike & Takagi, 2004) especially with the huge amounts of available data. On the basis of known sequences or structures, we can predict by homology those of a new protein. Hence, it enables the determination of the function and the classification of a query structure. Alignments methods are used to compare these biomolecules and to express their degree of similarity.

1.6.1 Sequential Alignment

A sequential alignment can be calculated between a pair of protein sequences called pairwise sequence alignment or concern more than two sequences named multiple sequence alignment

1.6.1.1 Pairwise sequence alignment

a) Model

Proteins can be modeled by a sequence of amino acids. Aligning proteins can be done by the comparison of their sequences: comparing symbols of amino acids sequences in order to identify a common ancestral sequence or to make functional or structural prediction. A

mutation may occur in the protein sequence. Therefore, to line up two proteins we should take into consideration these mutations by tolerating mismatches. Thus, an alignment of two amino acids occurs if they are the same or if one of them is substitutable by the other one. We can introduce a *gap* '-' when amino acids are removed because of mutations or make an insertion or deletion (InDel) to maximize the similarity between the sequences. These operations are called edit operations.

Example 1.7. *Let us consider the sequence : kasply.*
Then it can be edited into :

- *yasply through mutations*
- *kaspfly trough insertions*
- *kasp-ly trough deletions*

Example 1.8. *We consider two sequences : YRSRST and YMYSTR*

- *Sequence 1 : Y R S R S T*
- *Sequence 2 : Y M Y S T R*

A Possible alignment among them (edit operations are represented in boxes):

Y	R	-	S	R	S	T	-
Y	M	Y	S	-	-	T	R

b) Alignment measures

We have already seen, how we can model proteins to align them. In this part, we will present the measures to compare them in order to evaluate the quality of an alignment and then to find the optimal alignment.

Substitution among amino acids should be tolerated during the alignment and this because of mutations that may occur. However, a penalty is subtracted for mismatches and for each gap introduced because gaps increases uncertainty into the alignment. A constant is added if a match occurs. A score function is calculated according to the edit distance from the edit operations.

In order to find the alignment that maximizes the scoring system, matrices called similarity matrices are used. They contain distances between the different amino acids. Higher scores, i.e. distances are given to more-similar proteins, and lower or negative scores to dissimilar ones. We will use these matrices to compare protein sequences. We distinguish:

- Matrices based on the physicochemical characteristics of the amino-acids: Levitt matrix according to hydrophobicity (Levitt, 1976) and Levin matrix according to conformations (Levin et al., 1986).

- Matrices based on substitutions of the amino-acids during evolution: as Blosum[3](Henikoff, 1992) or as PAM[4](Dayhoff et al., 1978).

Example 1.9. *Let us continue with Example 1.8. We have this alignment :*

Y	*R*	-	*S*	*R*	*S*	*T*	-
Y	*M*	*Y*	*S*	-	-	*T*	*R*

Given these edit distances:

A match, it means a correspondence is scored +1, The mismatch between R and M according to a similarity matrix is scored -1 and the indel operation is scored -2. Then the score of this alignment is calculated as follows:

$$\begin{aligned} Score = \; & (numberofmatches) * Matchscore \\ & -(numberofmismatches) * Mismatchscore \\ & (numberofgaps) * Indel \\ Score = \; & (3 * (+1)) + (1 * (-1)) + (4 * (-2)) = -6 \end{aligned}$$

c) Type of Alignments

An alignment can be *local* or *global*. Local alignment compares only most similar and even non-contiguous portions of sequences whereas in global one the entire length of the sequences is compared and no amino acids is discarded. With sufficiently similar sequences, there is no difference between local and global alignments. However, global alignments are more appropriate for sequences that are similar and almost equal size; they attempt to align every residue in every sequence where local alignments are more suitable for dissimilar sequences that are suspected to contain regions of similarity. Two main and most used algorithms based on dynamic programming are the Needleman-Wunsch algorithm (Needleman & Wunsch, 1970) which is a general method for global alignment and the Smith-Waterman algorithm (Smith & Waterman, 1981) which is a general local alignment method.

Example 1.10. *Let us have two sequences :*
YTYYTEAVAAEXEX and YTEAAAEEX.
Using global and local alignments respectively, we obtain the following results:

Global:

Y	T	Y	T	E	A	V	A	A	E	X	E	X
Y	-	-	T	E	A	-	A	A	E	-	E	X

[3]BLOcks SUbstitution Matrix
[4]Point Accepted Matrix

Local:

Y	T	Y	T	E	A	V	A	A	-	E	X	E	X
-	-	Y	T	E	A	-	A	A	E	E	X	-	-

1.6.1.2 Multiple sequence alignment

As said by (Hubbard et al., 1996), "while pairwise alignment whispers multiple alignment shouts out loud". In fact, they are more sensitive in detecting relationships and patterns that are conserved among all proteins. It will serve for the identification and representation of a protein family, the detection of conserved regions and possible substitutions, and possibly recall evolutionary relationships by constructing phylogenetic trees. Thus, it is reliable because simultaneous comparisons of many molecules can reduce the noise and succeed to find relationships when pairwise alignment might fail specially when two sequences are not low similar. A consensus sequence which represents the most commonly amino acids found at each position of the sequences can be derived from multiple alignments to characterize a family of related molecules.

Multiple alignment is a computationally hard optimization problem which involves the consideration of different possible alignments in order to find an optimal one, given a measure of goodness of alignments.

For sequences, the sum-of-pairs (Carillo & Lipman, 1988) scoring is calculated as the sum of all pairwise alignment costs induced by it. Given two pairwise alignments x_i and x_j together with the induced alignment A_{ij} of x_i, x_j then the sum-of-pairs scoring is:

$$\mathrm{S(A)} = \sum_{i<j} \mathrm{S}(A_{ij}) \tag{1.1}$$

Example 1.11. *Let us suppose that among these three sequences : KASPY, ASPF and FY, we have found this multiple alignment which is a rectangular arrangement. Each row consists of one protein sequence padded by gaps. Thus, the columns highlight position conservation.*

K	*A*	*S*	*P*	-	*Y*
-	*A*	*S*	*P*	*F*	-
-	-	-	-	*F*	*Y*

To compute the score, we attribute 0 for a match and -1 for a mismatch.

K	A	S	P	-	Y
-	A	S	P	F	-

ScorePair1= -1+0+0+0+-1+-1= -3

K	A	S	P	-	Y
-	-	-	-	F	Y

ScorePair2= -1+-1+-1+-1+-1+0= -5

```
- A S P F -
- - - - F Y
```

ScorePair3= 0+-1+-1+-1+0+-1= -4

Then, according to Equation1.1, the sum-of-pairs scoring is calculated as follow:
SumPairs= -3 + -5 + -4 = -12

1.6.2 Structural Alignment

Protein structures are better conserved than sequences and similar structures may have similar functions. Thus, structural alignment is more informative though it is also more complex and difficult to analyze. The alignment should be geometrically and biologically significant.

1.6.2.1 Pairwise structural alignment

When searching for homologous proteins or for the purpose of classification, a pairwise alignment with a query structure can be made.

a) Model

The protein structure will be represented in a coordinate space. In fact, from a protein structure, features should be extracted. Techniques like geometric hashing, distance matrix (Holm & Sander, 1995), secondary structure elements(SSEs)(Artymiuk et al., 1994), side chain contacts, etc. are used to extract these features, i.e. structure descriptions. Protein structure is modeled with a graph that represents its folds and shape. A structural alignment is thus a graph alignment. A graph may represent for example protein atoms, c-alpha atoms or SSEs. An alignment is made by finding a best matching between the two graphs as shown in figure 1.11. Here the alignment obtained is:
{(p1,q1),(p2,q2),(p3,q3),(p4,q4),(p5,q5),(p6,q6),(p7,q7)}

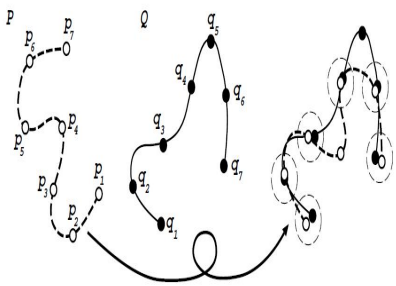


Figure 1.11: Alignment with Superposition

b) Alignment measure

As a measure to compare structural proteins representation, the Root Mean Square Deviation (RMSD) is computed as the square root of the *average of the squared distances* between

corresponding atoms of the two superposed structures. Thus, it computes the average atomic displacement between the two conformations. It indicates their divergence from one another. It is calculated as follows:

$$\text{RMSD}= \sqrt{\frac{1}{N}\sum_{i=1}^{N}\delta_i^2}$$

where δ represents the distance between the N pairs of equivalent atoms.

Its unit is Åangstöm (Å) which is equal to 10^{-10}m. The superposition with the lowest root-mean-square deviation (RMSD) is searched.

However, RMSD is a poor measure when the two shapes being compared differ significantly, particular when the two shapes contain some matching and some mismatching subshapes (Erdmann, 2005).

1.6.2.2 Multiple structural alignment

As for sequence multiple alignment, multiple structural alignment has an important role. In fact, it allows the identification of structural conserved regions and thus to build a consensus structure for protein families that can serve in the classification task.

a) Approaches

There are two approaches to make a multiple alignment. Their aim is to identify common features present in different molecules.

- **Template based approaches:** Many algorithms derive template from a family of related structures. For instance, we can mention PINTS (Stark et al., 2003) which detect patterns in non homologous tertiary structure. It compares a protein structure against a database of patterns to find similar spatial arrangements of amino acids in protein structures that are close in space but not necessarily adjacent in sequence. It is independent of sequence or overall fold similarity. Matches are evaluated by RAMS and a statistical E-value.

- **Comparison of individual structures:** Other algorithms focus on the comparison of individual structures. This can be done with the use of computer vision techniques to detect 3D structure motifs. Multiparty (Shatsky et al., 2002) multiple protein structural alignment algorithm, is an example of algorithm that compares individual structures to find common geometrical cores between the input molecules.

b) Sum-of-pairs distance

When we have to align several structures, we can use the sum of all pairwise squared distances (Sutcliffe et al., 1987) or we an use the average RMSD per aligned position (Ochagavia & Wodak, 2004).

For further details about algorithms used to make sequence and structural alignments, you can refer to Appendix A.

1.7 Conclusion

In this chapter, we have exposed the basics of bioinformatics: an interdisciplinary area that have for aims the storage, the organization, the analysis in a meaningful manner of the large amounts of biological data through the use of computational techniques and algorithms. We have presented some molecules that are studied in bioinformatics. Then, we have described different tasks that bioinformatics proposes to better understand and well manage biological information.

The comparison of proteins is one of the main problems of the molecular biology. Bioinformatics seems to facilitate this task with the use of computational methods and algorithms such as alignment methods to compare, in a linear way or in term of graphs, a pair or multiple, sequences and structures. To decide whether two molecule structures are related and exhibit their related regions, we have to compare graphs. The result can be used to characterize a functional family and thus predict the family of a query biomolecule. The ways to compare graphs and extract a similarity measure to make an alignment are described in the following chapter.

Chapter 2

Graph matching and similarity

2.1 Introduction

Protein structure is more closely to function than protein sequence. In fact, structural motifs may predict similar biological functions. Using graphs to model a protein is an appropriate way to represent relational information and to avoid its loss. In fact, proteins can occur in what is called *isomers* in the sense that they have identical composition and structural order but different configurations in three dimensions.

To align two proteins on a structural level, we consider each of them as a geometric object in the plane. These objects are represented by graph structures consisting of vertices (V), edges (E) connecting pairs of vertices, and attribute labels attached to vertices and edges. The graph matching problem is dealing with finding the best way to put into correspondence these vertices. In fact, to compare two graphs it is necessary to know which vertex of the first graph corresponds to which vertex of the second graph. One of the most intuitive ways to approach the graph matching problem is via the definition of a graph similarity measure.

Since the alignment is the process of lining up molecules to achieve a maximal level of identity, there is a need to identify region of similarity among graphs. The larger this common region is, the more precise and of better quality the alignment will be.

In this chapter, we are interested in different methods to express similarity and extract common regions on graphs. In Section 2.2 we will present methods that are used to extract similar regions and match graphs. Section 2.3 shows the complexity of these methods. A reformulation of graph matching by the mean of quasi-clique detection is detailed in Section 2.4. Section 2.5 describes the error tolerant graph matching method used for the alignment of multiple protein structures, called Multiple Graph Alignment.

2.2 Graph matching problem

Graph representations are widely used for dealing with structural information in different domains such as networks, system biology, pattern recognition. When similarity among graphs

are sought, the problem of graph matching have to be solved to find the best way to put into correspondence the graph vertices. There is two types of graph matching: The first one is the exact graph matching problem including graph isomorphism and subgraph isomorphism and the second one is the inexact graph matching problem called also edit graph distance problem. These problems will be detailed in the following.

2.2.1 Graph isomorphism

Two graphs are called isomorphs if they have the same number of vertices and if there is a bijective function between their sets of vertices. A graph is denoted G(V,E) where V represents the set of vertices and E the set of edges. Two graphs G1(V1,E1) and G2(V2,E2) are called isomorphic if:

- a function F exists:
  F: $V1 \rightarrow V2$, for any two vertices u,v in G1 such that:
  $(u, v) \in E1 \text{iff} (F(u), F(v)) \in E2$
- matched vertices have identical labels
- matched edges share the same labels

To determine whether two graphs are isomorphs or not, Ullman (Ullman, 1976) proposes to consider all possible node-to-node correspondences in a tree search space.

2.2.2 Subgraph isomorphism

A subgraph of G in a subset S of the vertices of G together with a subset of the edges connecting pairs of vertices in S. If a subgraph g1 of G1 is isomorphic to G2 than we are talking about subgraph isomorphism.

2.2.3 Maximum common subgraph

Given two graphs G1 and G2, G is called a common subgraph of G1, G2 if there is a subgraph isomorphism from G to G1 and from G to G2. The common subgraph is called maximum common subgraph (mcs) if there does not exist another common subgraph of G1 and G2 with more nodes than G.

2.2.4 Minimum common supergraph

G is called a common supergraph of two graphs G1 and G2, if there exist a subgraph isomorphism from G1 to G and from G2 to G. The common supergraph is called minimum common supergraph (MCS) if there does not exist an other supergraph with fewer nodes.

To assess the similarity between two graphs, the MCS and the mcs can be used to determine a similarity measure to compare graphs. For examples, the following metrics are used to compute the distance between two graphs G1 and G2:

$$\text{d(G1,G2)= 1-mcs(G1,G2)} \quad \max(\text{—G1—,—G2—}) \tag{2.1}$$

$$\text{d(G1,G2)= MCS(G1,G2)-mcs(G1,G2)} \tag{2.2}$$

2.2.5 Graph edit distance

Exact graph matching techniques are characterized by a stringent definition and a solid mathematical foundation. However the requirement that vertices and edges labels must match exactly is not applicable on graphs extracted from real-world data. Thus, error-tolerant graph matching (etgm) methods have been proposed. Their concept is to evaluate the similarity of two graphs in a broader sense (Neuhaus & Bunke, 2007). One of the most used methods for error-tolerant matching is graph edit distance.

The edit distance notion was originally proposed in the context of the string-to-string correction problem, where in (Levenshtein, 1966) a distortion model describing how two patterns differ from each other was defined. After its application on strings, edit distance was used on trees (Selkow, 1977) and finally on graphs (Bunke & Allermann, 1983). The graph edit distance concept allows us to measure the structural similarity of attributed graphs in an error-tolerant way.

It is a dissimilarity measure for arbitrary structured and arbitrary labeled graphs.It amounts to inexact graph matching techniques. In fact, we use an error model by allowing errors during matchings. Its basic idea is to define the dissimilarity of two graphs by the minimum amount of distortion that is needed to transform one graph into another. The graph edit distance is defined as:

$$\min \sum_{e_1,\cdots,e_n} c(e_i) \tag{2.3}$$

where c is a cost function used to quantify whether an edit operation modifies a graphs structure heavily or only slightly and e_i is an edit path between G1 and G2. We observe that this problem is a generalization of isomorphism. In fact, when the edit distance is equal to zero, G1 and G2 are isomorphic.

2.3 Complexity of graph matching

Graph matching is considered to be one of the most complex problems in object recognition in computer vision (Bienenstock, 1987). Its complexity is due to its combinatorial nature.

2.3.1 Complexity classes P, NP

2.3.1.1 Definitions used in Complexity theory

Polynomial time algorithm

A function f(n) is $\mathcal{O}$(g(n)) if there exists a constant c such that:

$$|f(n)| \leq c * |g(n)|, forallvaluesofn \geq 0$$

A polynomial time algorithm is defined to be one whose time complexity function is $\mathcal{O}$(p(n)) for some polynomial function p, where n is used to denote the input length.

Any algorithm whose time complexity function cannot be so bounded is called an exponential time algorithm. We shall refer to a problem as intractable if it is so hard that no polynomial time algorithm can possibly solve it.

Determinism of Turning machine

A deterministic machine is like a state machine which reads the input character by character and enters to the next state according to the current state and the read character. It is determined by the current state and the currently read character.

A non-deterministic Turing machine works otherwise like a deterministic machine, but it can have states where there are several alternative successor states, given the same input character. This means that the computation is not a direct path, but contains branches

2.3.1.2 Complexity classes P, NP

P-class: If a problem can be solved by a deterministic Turing machine in polynomial time, the problem belongs to the complexity class P.

NP-class: It contains all computational problems such that the corresponding decision problem can be solved in a polynomial time by a non-deterministic Turing machine.

- NP-complete: A problem is NP-complete if it is NP and all other problems in class NP can be reduced to problem p in polynomial time.

- NP-hard: NP-complete problems are a subset of NP-hard problems. In fact, all problems in class NP can be reduced to them. Very often, the NP-hard problems require exponential time or even worse.

2.3.2 Graph isomorphism

This problem is one of the few remaining problem in NP whose complexity is not yet been classified within a particular type being either P or NP-complete. Some papers in the literature tried to prove its NP-completeness when the two graphs to be matched are of particular types or satisfy some particular constraints, but it still remains to be proved that the complexity of the whole type remains within the NP-completeness. On the other hand, for some types of graphs the complexity of the graph isomorphism problem has been proved to be of polynomial type. For instance, planar graphs have been proven in (Hopcroft & Wong, 1974) to be of polynomial complexity.

2.3.3 Subgraph isomorphism

It has been proven to be NP-complete (Garey & Johnson, 1979). However, some specific types of graphs can also have a lower complexity. For instance, the particular case in which the big graph is a forest and the small one to be matched is a tree has been shown to be of a polynomial complexity (Garey & Johnson, 1979).

2.3.4 Error-tolerant graph matching

The computational complexity is proved to be NP-complete. In fact, the number of possible matches is correlated with the number of nodes of the involved graphs.

2.4 Graph matching reformulation: Clique-detection

2.4.1 Motivation

The problem of finding a common subgraph can be reduced to an other graph theory problem: the clique-detection problem that belong to the same complexity type (NP-complete) and for which many algorithms have been proposed. Levi (Levi, 1972) has shown that the mcs corresponds to a maximum complete subgraph (a clique) in a so-called product or association graph. This graph is noticed A(V,E) where V $\subseteq V_1$ X V_2. A contains a node for each pair of nodes from the input graphs that are compatible with each other as they share the same node label and an edge between two nodes are inserted if both the two graph G_1 vertices and the two graph G_2 vertices are connected and share same weights or if neither the two graph G_1 vertices nor the two graph G_2 vertices are connected.

2.4.2 Clique definition

A clique is a graph G is a subset of vertices in the graph such that each pair of vertices in this subset is connected by an edge. It is also named a complete subgraph.

- Maximum clique: It is the largest subset in a graph such this subset is a clique.
- Maximal clique: It is a clique that is not contained in any other complete subgraph. Among all maximal cliques, the largest one is the maximum clique.

2.4.3 Clique detection

To detect cliques many algorithms exists. Some of them focus on finding maximal cliques(clique-enumeration problem), others seek for only the maximum clique.

The Bron-Kerbosch algorithm (Bron & Kerbosch, 1973) is the most widely used algorithm because it appears to be the most efficient and fastest algorithm in most of the cases for enumerating all maximal cliques. It uses a backtracking method on a recursive tree search, i.e. if no extension is possible from the building a the path from vertex v_1 to vertex v_k than delete v_k and continue with extending the path $v_1..v_{k-1}$).

Example 2.1. *Given two graphs G1 and G2, we have to produce a product graph P on which we search for maximal cliques. In Figure 2.1, the maximal cliques found, i.e. represented by the set of nodes 1,2,3 as well as 1,2,4,5, are the common substructures of the graph G1 and the graph G2.*

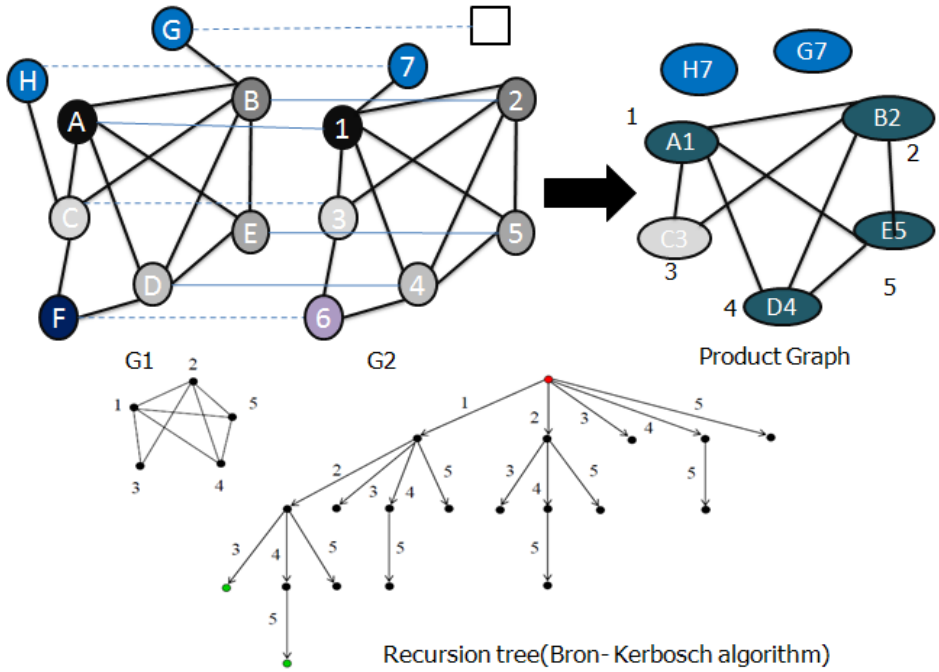


Figure 2.1: Clique detection for mcs problem

The problem of graph matching can be reformulated into others problems, for instance into a constraint satisfaction problem.

2.5 An Error-tolerant graph matching method for the multiple structural alignment

2.5.1 Motivation

The graph-based similarity between the graphs representing molecules plays an important role in many aspects of chemistry and systems biology.

In this section, we present the Multiple Graph Alignment method (MGA) (Weskamp et al., 2007). It focuses on the structural analysis of protein binding sites. As its name indicates, it analyzes more than two graphs simultaneously and seeks for an alignment among them. Graphs are used to represent protein's binding site, comparing them amounts to graph matching. It uses inexact graph matching techniques among several graphs and identify templates. These templates can serve to classify new structures. Thus, it will allow the identification of not only highly conserved regions among graphs to characterize a functional family but also weakly conserved ones, i.e.that are not found in each member of the family. The MGA method allow also the detection of discriminative features among related families. These features are useful specially in the pharmaceutical field where we need them to design a specific drug.

Graphs contain information about the shape and the physicochemical descriptors of a protein, they will be aligned and by this way this method allows the identification and characterization of functional relationships among proteins independently of sequence or fold

homology. Ignoring these homologies is due to the fact that many molecules though have no sequence homology, know a resemblance on 3D structural level, e.g. between myoglobin and hemoglobin and they are functionally similar and are involved in the storage and the transport of oxygen. Besides, in the cavity space mutations may occur and can result in the replacement of certain functional groups by other functional groups while the overall fold is preserved, that is why comparing folds can lead in error. Thus, MGA method is considered as a non-homology based method. In fact, non-homologous molecules with similar functions will be captured through the alignment.

2.5.2 Representation of graphs

For each binding pocket, pseudocenters representing its geometrical and physicochemical properties are identified using rules defined in (Schmitt et al., 2002). These pseudocenters will be represented through a connected, node-labeled and edge-weighted graph G(V,E) where node labels $l(v)=1,\ldots,7$ stands respectively to physicochemical properties (hydrogen-bond donor (DO), Acceptor (A), Mixed donor/acceptor (DA), Hydrophobic aliphatic (AL) and aromatic (Ar), π-π interaction (PI) and Ion metal (IM)) (Schmitt et al., 2002). Two centers are connected if their Euclidean distance is below a threshold of 11Å[1] which is applied to simplify the graph representation and at the same time rise constraint for the node connection, i.e. longer edges actually exist but they are ignored. This distance is considered as enough to capture the geometry of a binding site. The edge $e \in E_n$ will be labeled by the distance separating them $w(e) \in \mathbb{R}$. Thus, the alignment sought is a one-to-one mapping of vertices of different graphs onto each other, in a way that the correspondence of vertex labels as well as the mapping of edges with similar distances onto each other is maximized. Therefore we expect the graphs to be vertex labeled and edge weighted, where edge labels are interpreted as distance values.

2.5.3 Construction of MGA

Multiple graph alignment concept is seen as a structural counterpart to multiple sequence alignment (Weskamp et al., 2007). As a result of protein flexibility in space or because of mutations, the binding pocket's spatial structure and so the graph descriptors will be affected and distorted. Consequently, we can find mismatches between two related proteins' graph.

2.5.3.1 Edit operations

Edit operations, and therefore the concept of errors into matching is used to explain the transformation of one graph to another one. Insertion or deletion of a node, a change of the node label or edge's weight are thus taken into consideration. A dummy node ($\perp$) is used as for a gap in sequence alignment. Hence, between two graphs we can have:

- a node match(n_m) if $l(a^i_j)$=l(a^i_k)
- a node mismatch (n_{mm}) if $l(a^i_j) \neq$ l(a^i_k)
- a node dummy (n_d) if there are an insertion or deletion

[1] 1 Angström (Å)=10^{-10}meter

- an edge match (e_m) if $d_{kl}^{ij} \leq \epsilon$
- *an edge mismatch(e_{mm}) otherwise, i.e. an edge that exist either in the first or second graph, or a difference superior to ϵ between edge's weights was found.*

However, saying that two edges match the difference between their lengths is less than ϵ is too restrictive. Thus, this concept of edge matching was extended by means of fuzzy subsets of the distance values d $\in$ [0,∞[, namely by two fuzzy intervals M,$\bar{M}$: [0,∞[$\rightarrow$ [0,1] (Hüllermeier et al., 2007). Therefore, two edges can match and mismatch according to a certain degree. There are linear transition region around d = 2 (the width of the transition region is 1, i.e., a distance ≤ 1 is regarded as a perfect match, while a distance ≥ 3 is a full mismatch.

A scoring function is computed. In fact, Each operation has a *cost function* which will be summed at the end into node score (ns) for nodes' operations and edge score (es) for edges' operations. With the extension of edge matchings, es is also extend. Two cases of edges scores can be encountered: a missed edge in one of the two graphs which is considered as es_{dummy} and otherwise es_m M(d_{kl}^{ij}) + es_m $\bar{M}(d_{kl}^{ij})$.

So, two graph descriptors have an *edit distance* which is calculated as the cost of cost-minimal sequence of edit operations necessary to the transformation. This score assess the quality of the alignment.

A multiple alignment among graphs $A \subseteq (V_1 \cup \perp) \times \ldots \times (V_n \cup \perp)$ should take into account that:

- each node of each graph must occur exactly once in the alignment:
for all i = 1,...,n and for each $v \in V_i$ there exists exactly a = $(a_1,\ldots,a_n) \in A$, such that $v = a_i$) and

- each tuple of the alignment must contain at least one non-dummy node:
for each a=$(a_1,\ldots,a_n) \in A$ there exists at least one 1$\leq$i$\leq$n, such that $a^i \neq \perp$.

2.5.3.2 Used algorithms by MGA method to make a multiple alignment

Greedy strategy

This strategy is composed of two steps. It starts with a series of pairwise alignment in a first step and then combine these alignments in an incremental manner to obtain a multiple alignment in a second step.

Pairwise alignment step: To make a pairwise graph alignment between two graphs $G_1(V_1,E_1)$ and $G_2(V_2,E_2)$, the maximum subgraph isomorphism between G1 and G2 should be found , i.e. the maximum common subgraph (mcs) which provides a measure of structural similarity. For that, clique detection was applied. Once a common substructure is found, remaining nodes that do not belong to it have to be added gradually, allowing mismatches.

Example 2.2. *Given two graphs, G1 and G2, from which we extract the maximum common substructure using the Bron-kerbosch algorithm on their product graph. Once, this substructure*

*is defined,'A1-B2-D4-5E' in Figure2.2, it will be used as a seed solution for further extension to include remaining nodes of G1 and G2. The first node not included yet in the solution and should be treated at first is that has the highest number of edges to those nodes already part of the solution. In Figure***??** *node C which will be mapped onto node 4. The process continues until all nodes are matched together. Though different combinations are possible, the one that maximize the score will be chosen.*

Incremental combination of multiple pairwise alignments step: Here, MGA can be seen as pairwise graph alignments applied several times in an incremental manner. In fact, during the pairwise alignment step, all graphs were aligned to a center structure called pivot. Then, in this step, they will be merged together according to this pivot structure. This greedy strategy is called star alignment.

Example 2.3. *Suppose that we have a three graph, G1,G2,G3, and the pivot structure is graph G2. From a pair of pairwise alignment: G1, G2 and G3, G2, we want to have an alignment of G1,G2,G3. Let us continue with the first alignment presented in Example 2.2 to make a multiple alignment. The resulting alignment is represented in Figure 2.3.*

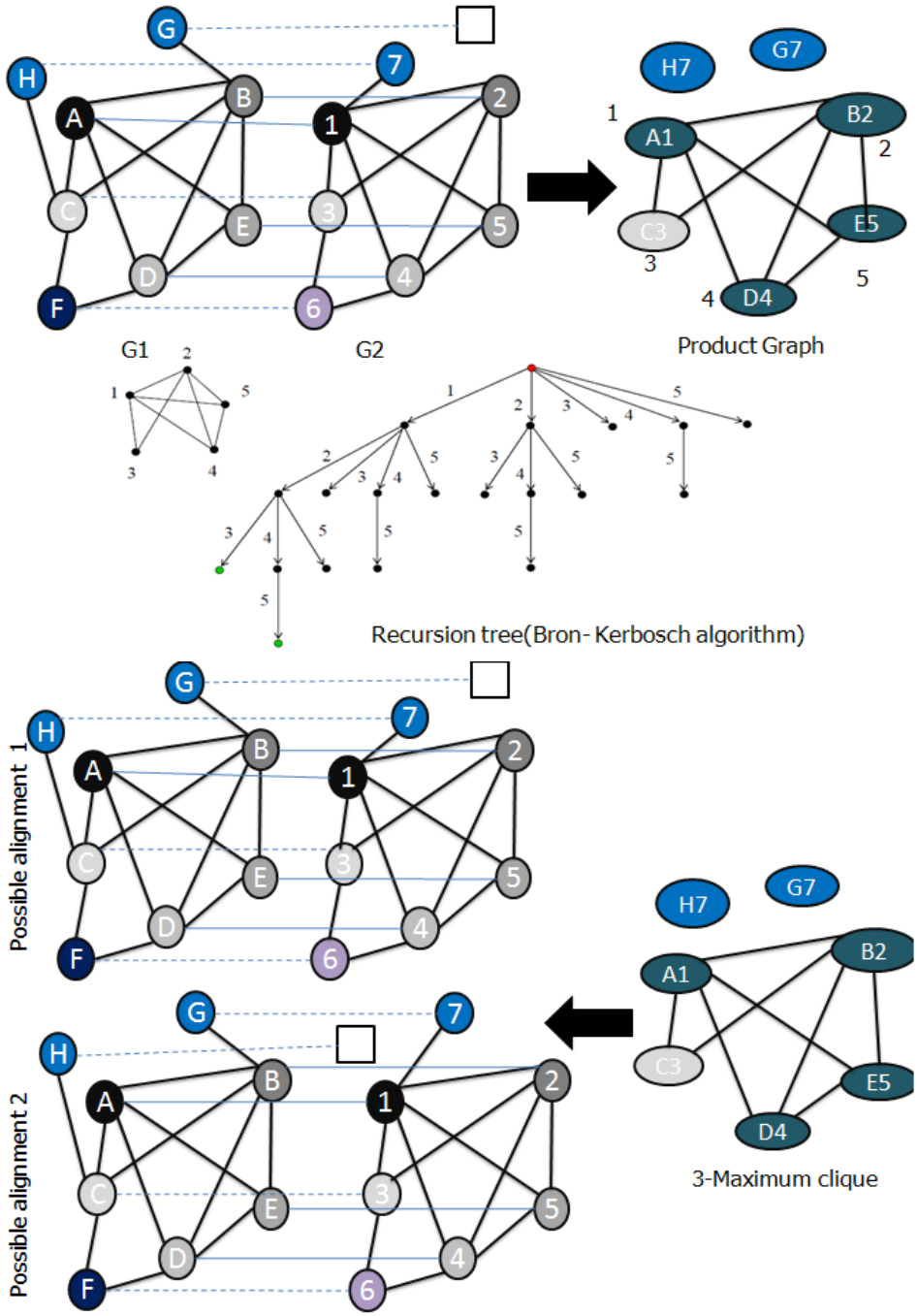


Figure 2.2: Pairwise graph alignment

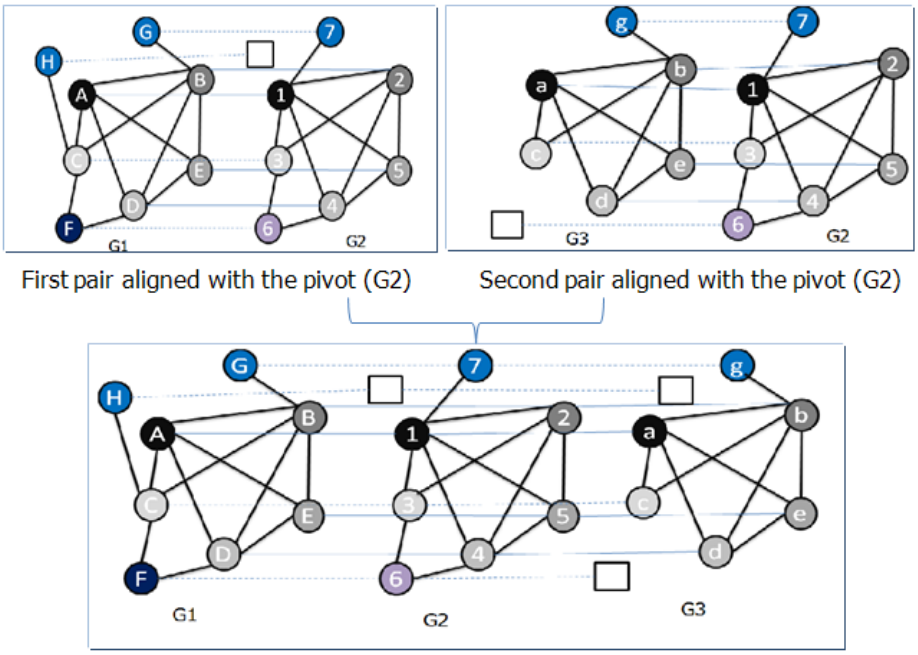


Figure 2.3: Multiple graph alignment

The weak point of this method is the choice of the pivot because the quality of the alignment depends on it and in order to improve this quality many structure should be tested as pivot before fixing the optimal one, whereas its advantage is that any new graph descriptor that should be added to the alignment will be easily inserted by just aligning it with the pivot. In fact, multiple alignment is seen as an incremental combination of multiple pairwise alignments. The goal is to optimize the graph alignment by maximizing the score function of the MGA A= a^i,...,a^n. It is based on the sum-of-pairs scheme, i.e. the score of the multiple alignment is reduced to a series of scores of pairwise cases:

$$\mathrm{s}(A)=\sum_{i=1}^{n} \mathrm{ns}(a^i)+ \sum_{1<i<j<n} \mathrm{es}(a^i,a^j) \tag{2.4}$$

b) Evolutionary strategy

Evolutionary Algorithms (EA)

Finding an optimal alignment among several graphs is always sought. An alignment is assessed by its reliability, its time of execution and its quality which is indicated by the score. Here, evolutionary algorithms (EA) are used to align multiple graphs as an alternative strategy (Fober et al., 2008). Even if evolutionary optimization is more expensive from a computational point of view, this algorithm is used to improve the solution quality, i.e., to produce alignments that are better than those obtained by the simple greedy strategy.

It is based on the *evolution process* of a family of solutions. These solutions will be mutated and recombined until optimal solutions satisfying different stopping criteria. Fitness is measured according to equation 2.4).

At the beginning, solutions (MGAs) are randomly generated, will represent individuals. These individuals will be *combined* and will generate an offspring solution which can be mutated. Then, this offspring solution will be added to the parent population and may be selected to be in the population of the next generation. A selective measure (the deterministic plus-selective operator) is used for this purpose. In fact, only best solutions are kept to form the next population. Hence, we draw nearer progressively the optimal solution.

In this case, an individual represents an MGA by a matrix where rows represent graphs and columns aligned nodes (see Figure 2.4). The problem here is how to fix the number of columns in order to reduce the search space and at the same time do not loose the optimal solution. As a solution, we start with n+1 columns (n: max graphs' node number, +1: for dummies) and add other columns if there is a need.

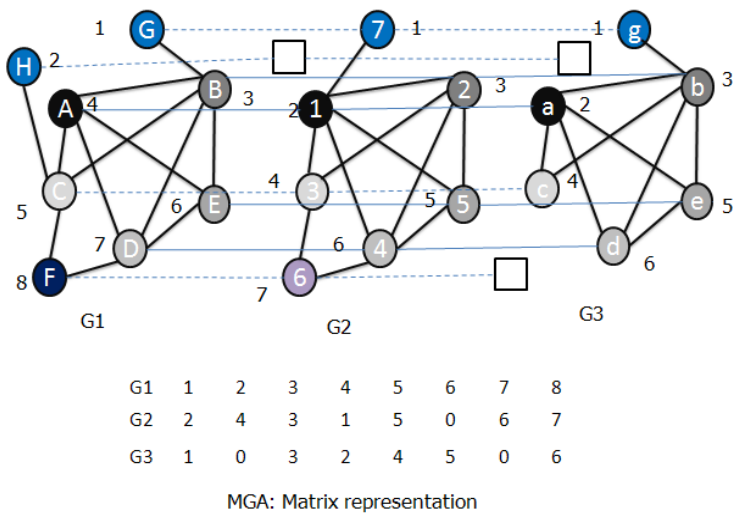


Figure 2.4: Matrix representation of MGA and its corresponding graph alignment

Sub-matrix will be combined and the offspring matrix will be constructed. Mutation is represented by a cell's swap of one row and two columns. This method, even if it shows a higher computational complexity, improves the quality of the alignment.

Example 2.4. *Let's suppose that we have two candidate solutions (MGAs),MGA1 and MGA2, represented by two matrices (Figure 2.5). We must take a pivot graph to arrange the columns of the remaining graphs according to the order of its columns. In this case, if we take the first graph as a pivot, the offspring solution will be:*

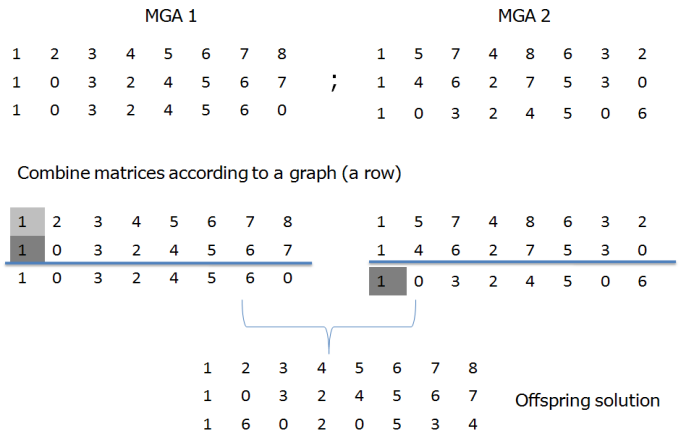


Figure 2.5: Generation of an offspring solution

A Variant of EA (EA*)

EA increases the alignment quality, though its runtime is very high when it is applied on large structures. It was noticed that when it is applied on a pair of proteins, it give good results and a shorter time of execution. Thus, combining evolutionary optimization with pairwise decomposition seems to be a good idea. The EA* is a variant of EA, since it uses EA to solve pairwise alignments and applies star-alignments to combine them into a multiple alignment. Even if there is a slight decrease in the solution quality compared to the evolutionary one, it still a good one and complexity was reduced.

2.5.4 Consensus graphs

What differentiate a multiple alignment from a pairwise alignment is the fact that it allow the identification of consensuses. As for sequences where a consensus sequence is represented by amino acids most commonly found at each position of the sequences of related molecules, an MGA allow the derivation of a consensus graph. Hence, a consensus graph is a graph that characterize a functional family class. It is represented by conserved (not necessarily highly and perfectly) pseudocenters across aligned binding pocket, its means that a node is represented in this graph if it is found in at least one of the aligned graphs and by this way also weakly conserved patterns are identified. In this method, degrees of conservation are specified which can even be used if there are a need to limit the size of the consensus graph.

It is interesting to determine which of the pseudocenters are highly conserved across all aligned binding pockets. In fact, these pseudocenters can be regarded as characteristic for the respective functional class.

In order to identify these pseudocenters, the "highly conservation concept" where a label is present in most of the aligned structures was expressed as a fuzzy property. Thus, a fuzzy consensus graph is derived from the alignment. Nodes are labeled with the degree of conservation of each physicochemical property (from 1 to 7 where 1 refers to donor, 2 acceptor,$\cdots$).

Hence, each node will be labeled in a fuzzy way by a vector :

$$\mathrm{l(v')} = (\mathrm{con(v',1)},\mathrm{con(v',2)},\ldots,\mathrm{con(v',5)}) \tag{2.5}$$

This requires a formal specification of the "For Most" quantifier, which is considered as a relaxation of the universal quantifier. It is defined as an increasing function of relative frequency:

$$\mathrm{con(v',l)} = \mathrm{ForMost}(\tfrac{1}{n}\sum_{i=1}^{n}\mathbb{I}(a_i{=}l)) \tag{2.6}$$

where ForMost(·) is a function such as

$$\mathrm{ForMost(x)} = \max\ (0,\min(3\mathrm{x}\text{-}1.8,1)) \tag{2.7}$$

Example 2.5. *In this example a consensus graph with fuzzy node labeling is build. We can see in Figure 2.6 that the first node is highly conserved while the last node is weakly preserved.*

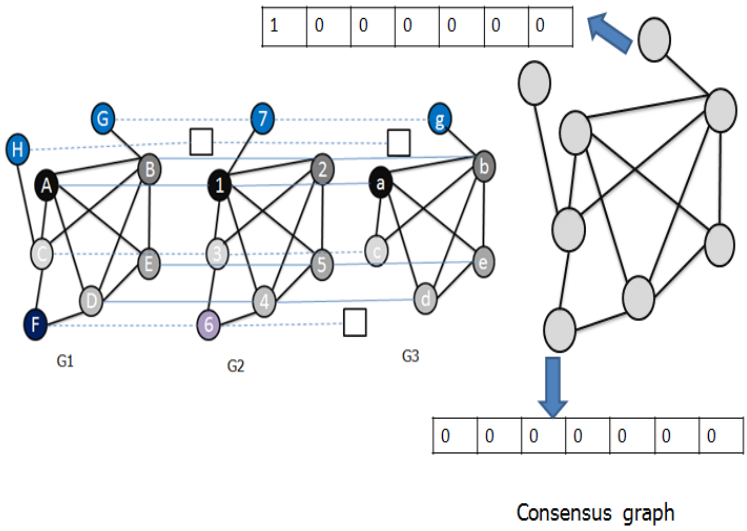


Figure 2.6: Fuzzy consensus graph

Edges in the consensus graph must be weighted by a *prototypical length*. Thus, a degree to which a length x∈[0,11] is prototypical is defined as the degree to which the length of most edges in the aligned structures is close to x. As for nodes, a formalization of the Formost quantifier is also needed. Besides, a fuzzy representation of "being close to x" have to be used. Thus, for each edge a matching degree with an x value is calculated according to the fuzzy set M seen above the recording of match degrees is denoted as $\mu^{(1)},\ldots,\mu^{(n)}$ with $\mu^{(1)}\geq\ldots\geq\mu^{(n)}$. The degree to which a length x is prototypical for an edge e is given by :

$$\mu_e(\mathrm{x}) = max_{k=1..n}\min(\mathrm{ForMost}(\ \tfrac{k}{n}\ ,\ \mu^{(k)})) \tag{2.8}$$

2.5.5 Classification

A consensus graph characterizes a functional class of protein structure and therefore can be seen as a prototype for this family. Thus, it can be used to make classification. In fact, a

consensus graph can be considered as prototype for *k-nearest neighbor classifier* where the score of the pairwise alignment between a query structure and different consensus represent the different distances. The k-nearest distances are picked and then the result of a majority vote between the functional family of the k-consensus selected, will predict the functional family of the query structure.

A fuzzy consensus graph can be computed as shown above, where the consensus structure has a degree of conservation for each node and edge. This structure represents better a functional family than a single non-fuzzy graph because of these degrees of conservation(i.e. even weakly conserved pseudocenters are taken into consideration). Thus, it is appropriate to use it to make classification.

In this case, a query structure will be aligned with different consensus graphs. Each time, a degree of membership to the class modeled with the current consensus graph will be computed.

Example 2.6. *Given consensus graphs where each of them is representative of a functional family, and a query structure with an unknown family. The aim is to predict, by similarity to the known structures, the functional class of this query structure. Figure2.7, shows that the score of pairwise alignments between the query structure and each consensus will be used as a distance. A majority vote between the functional class of the k-biggest (a negative score is given for mismatches) distances will indicate the class of the query structure.*

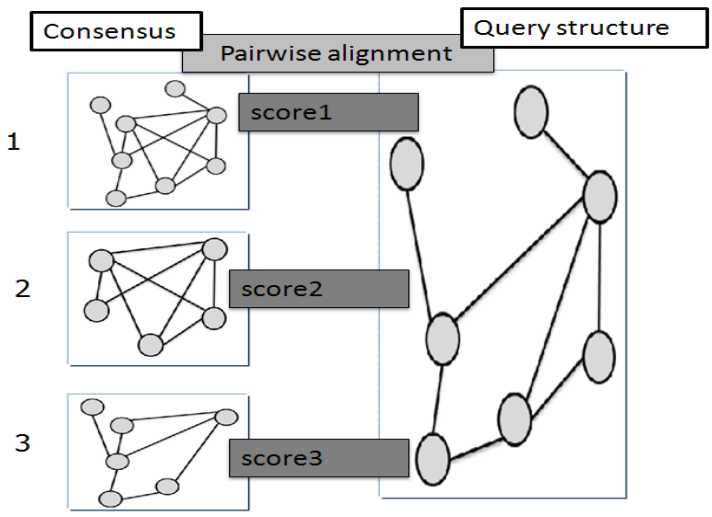


Figure 2.7: Classification

To score the pairwise alignment between the query structure and a consensus graph, nodes match and mismatch are defined as:

$$\text{match(u',v')} = \bigoplus_{i=1}^{5} \text{con(u',i)} \otimes \text{con(v',i)} \tag{2.9}$$

$$\text{mismatch(u',v')} = \bigoplus \text{con(u',}l_1\text{)} \otimes \text{con(v',}l_2\text{) where } 1 \leq l_1 \neq l_2 \leq 5 \tag{2.10}$$

where $\otimes$ is a t-norm (min is used) and $\oplus$ a t-conorm (max is used), then nodes are scored:

$$\text{ns}= ns_m \text{ match(u',v')} + ns_m m \text{ mismatch(u',v')} \tag{2.11}$$

Similar to the nodes the match and mismatch between two prototypical weight of edges x and y is computed according to:

$$\text{match}(e_1,e_2)= max_{xy}\mu_e 1(\text{x})\otimes\mu_e 1(\text{y})\otimes \text{ M}(\|\text{x-y}\|) \tag{2.12}$$

$$\text{mismatch}(e_1,e_2)= max_{xy}\mu_e 1(\text{x})\otimes\mu_e 1(\text{y})\otimes \bar{M}(\|\text{x-y}\|) \tag{2.13}$$

and then

$$\text{es}=es_m \text{ match}(e_1,e_2) + es_m m \text{ mismatch}(e_1,e_2) \tag{2.14}$$

2.6 Conclusion

Graph matching is used when there is a need to determine correspondences between the vertices and edges of attributed structures. In this chapter, we have seen that, thought exact graph matching techniques like graph and subgraph isomorphism are very rigorous, they are applicable only on few real world data because they require that the structures to be matched are identical. For that reason, Error-tolerant graph matching techniques were proposed. In fact, the identity notion is relaxed to similarity. After that, we have presented an error-tolerant method that seeks to match multiple graphs and derive a structural multiple alignment.

Several researches are trying to draw nearer to the optimal alignment. It was shown that the way by which an alignment is made affects the quality of the results. When the common substructure is large, the number of inexact matchings will be reduced and therefore the alignment quality will be better. The idea in this master thesis is to widen the similarity regions already detected by clique-detection by the mean of a relaxed concept: quasi-cliques and thus improve the comparison of graphs. The quasi-clique concept and how it will serve to measure the similarity between graphs will be detailed in the following part of this report.

Part II

Graph-similarity based on quasi-clique detection

Chapter 3

Graph-similarity based on quasi-clique detection

3.1 Introduction

As mentioned in Chapter 2, the representation of chemical and biological molecules by means of graphs allows their structural analysis. The relationships between graphs are detected using graph matching algorithms. The graph matching problem can be formulated as a problem involving the maximum common substructure (mcs) between the collection of graphs being considered and the clique detection algorithms were proved to provide an efficient way of implementing mcs detection.

Cliques are the densest form of subgraphs. In fact, there is an obligation to have an edge between each pairs of vertices. This requirement is too restrictive given that real-world data sets are often incomplete and noisy(Liu & Wong, 2008).

Because of the flexible nature of the protein, distortions can be observed on the protein 3D-structure. Thus, the spatial arrangement of the pseudocenters of the protein binding pockets will be affected. Since these pseudocenters are represented with graph nodes, the coordinates of these nodes will be modified and the original distance between two nodes will therefore be distorted. As explained in Chapter 2, the edges in the binding pocket graph are weighted by this distance (under 11Å) between pseudocenters. Thus, distortions will concern edges. Indeed, an edge can be present in a first graph and absent in the second graph because of a spatial conformation, while these two graphs are similar in the linear plane.

Applying the clique concept, will penalize such distortions and restrain the common substructure to the nodes linked by exactly the same number of edges in both graphs being compared. To optimize the size of the common substructure, we will relax the exigency of cliques by using quasi-cliques where a node should be only linked to a specified number of other nodes and not necessarily to all of them. This relaxation will identify similar structures even if they have different conformations.

This chapter is consecrated to the presentation of our manner to measure the similarity

between graphs through the discovery of quasi-cliques. Since our work is based on the quasi-clique concept, we expose its definition in Section 3.2. Section 3.3 details maximal quasi-cliques detection problem. In Section 3.4, we explain the way we use the quasi-clique concept as a measure of similarity between graphs.

3.2 Quasi-clique basics

In this section, we expose the definitions of the quasi-clique notion and the different notations that we will use to handle it.

3.2.1 Notations

The following notations will be used:

- For a graph G(V,E), V: the set of vertices; E: the set of edges. Two vertices are adjacent, if there is an edge between them.
- γ: a user-specified number between 0 and 1 called the minimum degree threshold
- $N^G(v)$: the adjacency list of a vertex V, it is the set of vertices u as $(u,v)\in$ E.
- $N_k^G(v)$: the set of vertices u that are connected to (v) with a distance less or equal to k, $N_k^G(v)= (v)|d(v,u)\leq k$ for $k\geq 1$.
- $deg^G(v)$: a vertex degree, it represents the cardinality of $N^G(v)$.
- *diam*(G): the diameter of a graph, it is the longest shortest path in the graph G : *diam*(G)= $max_{u,v\in V(G)}$ $d(v,u)$.
- *cand_exts(X)*: the set of vertices that may extend the set of vertices X. They are the vertices after the last vertex of X.
- $indeg^X(u)$: the number of vertices in X that are adjacent to u.
- $exdeg^X(u)$: the number of vertices in *cand_exts(X)* that are adjacent to u.

3.2.2 Definitions

Quasi-cliques are almost complete graphs. Different definitions have been given to quasi-cliques. Some of them are based on the degree of the vertices (Pei et al., 2005), (Matsuda et al., 1999) while others deal with the edge density (Abello et al., 2002).

- A graph or more precisely a subset of vertices X of a graph is mentioned as a quasi-clique if every vertex in X is adjacent to at least [γ(n-1)] other vertices, n is the number of vertices in X.
- A graph G(V,E) is a quasi-clique if |E|= $\gamma \binom{|V|}{2}$; γ in [0-1]

We notice that quasi-cliques are a generation of cliques. In fact, if γ is equal to one, the γ-quasi-clique is a clique. The degree of a vertex must be an integer and a γ-quasi-clique G must be connected. Hence, $deg^G(v) \geq \gamma$(n-1) and therefore $\gamma \geq \frac{1}{n-1}$.

A γ-quasi-clique on a vertex set X is maximal, if there is not a vertex set Y such that X $\subset$ Y and an induced graph on Y is a γ-quasi-clique.

Example 3.1. *Given a graph G, where each vertex* v*, in the set of vertices V, has a degree equal to two (see Figure 3.1).*

If γ is equal to 0.5 then G1 will be a 0.5-quasi-clique since the requirement that deg^G(v)$\geq \gamma$(n-1) is satisfied: 2$\geq$2.*
Let us check if G_1 is a 0.6-quasi-clique:
If γ is equal to 0.6, the degree of each node in G1 should be at least equal to 0.6(5-1)= 2.4*
However, we can see that G_1 can not be a 0.6-quasi-clique because the degrees of nodes is equal to two and 2$\leq$2.4.

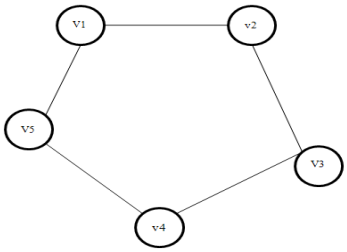


Figure 3.1: γ-quasi-clique

3.3 Quasi-clique detection: QCD

3.3.1 Difference with clique detection

Discovering maximal cliques is a hard task since there is a correlation between the number of nodes in the graph and the spatial and runtime complexity. To prune the search space, the downward closure property is used:

If a graph G is a clique, for any supergraph G' as G $\subset$ G', G' must be a clique. Thus, if a subset is not a clique, all its supergraphs will not be cliques.

This downward closure property does not hold for quasi-cliques. In fact, if a graph is not a γ-quasi-clique, it does not necessarily mean that its supergraphs are not also quasi-cliques.

Example 3.2. *If we consider a 0.5-quasi-clique (see Figure 3.2(A)), we show an example of one of its induced subgraph (see Figure 3.2(B)) which does not form a 0.5-quasi-clique.*

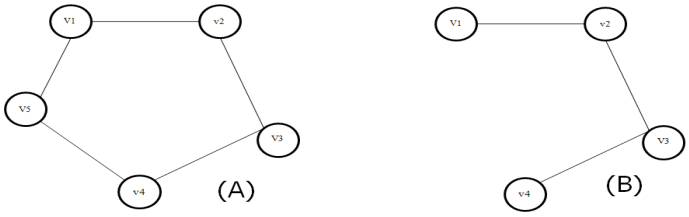


Figure 3.2: A quasi-clique and its subgraph

3.3.2 Quasi-clique detection

On a single graph G, discovering quasi-cliques in G get back to find all the subsets X such that an induced subgraph on X is a maximal γ-quasi-clique.

Since, the downward property does not hold, any subset of the set of vertices V can form a γ-quasi-clique. All potential γ-quasi-cliques can be represented with a set-enumeration tree (Rymond, 1992) where nodes' tree represent subsets of the power set of the set of vertices V.

Example 3.3. *Given a graph of four nodes: a,b,c,d and an order on the nodes such that $a \geq b \geq c \geq d$. Figure 3.3 represents its corresponding set-enumeration tree.*

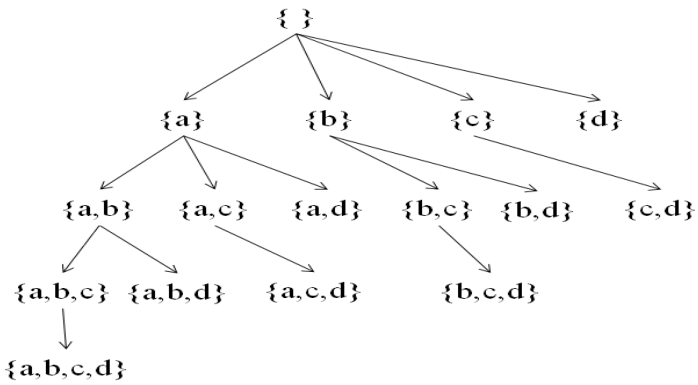


Figure 3.3: Set enumeration tree corresponding to the search space

The set of quasi-cliques is found by making a depth first search on the set-enumeration tree and checking if the vertex sets are quasi-cliques. Once quasi-cliques are discovered, Liu et al. (Liu & Wong, 2008) have proposed to store them in a prefix-tree in order to determine maximal ones. Hence, some leaves in the prefix-tree will represent maximal γ-quasi-cliques.

3.3.3 Pruning techniques

Since as explained before, the closure property does not hold for quasi-cliques, mining them is more expensive than detecting maximal cliques. Indeed, we have to explore in a depth-first search the set enumeration tree of all the nodes which size is exponential to the number of vertices in the graph. Thus, pruning techniques were proposed to reduce the search space and therefore reduce the runtime complexity. In our work, to mine quasi-cliques, it seems

appropriate for us to use these pruning techniques. In the following, we will present some of these techniques based on:

- Minimum size degree: In order to eliminate trivial quasi-cliques, a minimum size of the number of vertices in the quasi-clique can be fixed.
- Diameter of the graph: Pei et al. (Pei et al., 2005) have proved that there is a relationship between γ and *diam*(G). For instance, if $\gamma \geq 0.5$ then *diam*(G) should be less or equal to two.
- A lookahead technique: In (Liu & Wong, 2008), there is a notice that if X $\cup$ *cand_exts(X)* is a γ-quasi-clique then there is no need to extend X further because the maximal quasi-clique will be X $\cup$ *cand_exts(X)*.
- The degree of the vertices: Zeng et al. (Zeng et al., 2006) have proposed to prune the search space according to the number of neighbors of X according to $indeg^X(u)$ and to $exdeg^X(u)$.
  - $indeg^X(u) + exdeg^X(u) < \gamma(|X|+exdeg^X(u)$
  - $indeg^X(u) < \gamma|X|$ and $exdeg^X(u)=0$
  - $indeg^X(u) + exdeg^X(u) < \gamma(|X|-1+exdeg^X(u))$
- The upper bound U, the maximum number of vertices that can be added to X to form a y-quasi-clique
- The lower bound L, the number of vertices that can be added to X to increase the degree of the vertices in the current subset to form a y-quasi-clique
- Critical vertices: a vertex u is considered as critical if: $indeg^X(u) + exdeg^X(u) = \gamma(|X|-1+L_X)$. If such vertices exist, Liu et al have shown that the extension of X should take into consideration only vertices that are adjacent to critical vertices to X (Liu & Wong, 2008).
- Cover vertices: if u is a cover vertex, then X will be extended only by the vertices in *cand_exts(X)* from which we remove the set of vertices that are covered by u.

3.4 Graph similarity based on QCD

3.4.1 Motivation

Many graph-based methods measure the similarity between two graphs by the maximum common subgraph (Raymond & Willet, 2002), (Hattori & Kanehisa, 2003). They measure the similarity between two graphs by calculating the size (number of nodes) of their common substructure.

We explore the concept of quasi-cliques as a measure of similarity between graphs. This relaxation of cliques, will help to produce a larger common substructure between the input graphs. Figure 3.4 shows the effect of the use of quasi-clique detection instead of clique detection on the size of the common substructure.

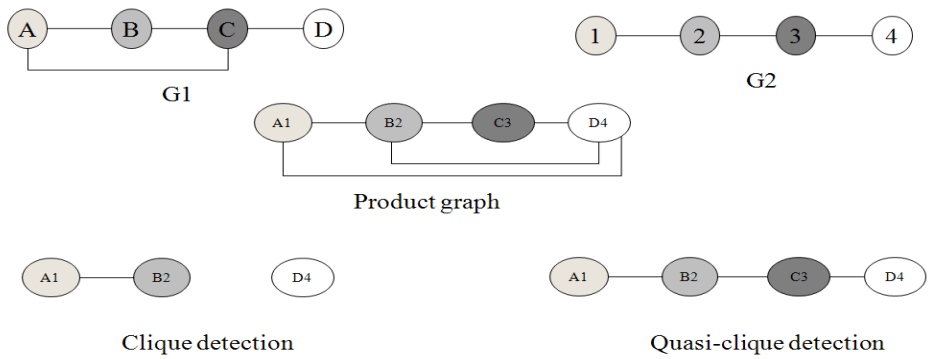


Figure 3.4: Difference between clique and quasi-clique measure

3.4.2 Objective

The search of protein similarity we handle is completely independent of sequence and fold information. Comparing protein structure amounts at measuring the similarity between structures as large mcs. However, it does not guarantee of meaningful substructures and therefore it will affect the classification task. For instance, a large subgraph whose vertices are separated is of little meaning while a smaller connected subgraph may be much meaningful. The major objectives of this work are:

- The construction of a modified product graph from two input graphs to decrease the number of false-positive matches (vertices sharing the same node label just by chance) and trivial common substructures.

- The Development of a graph similarity measure based on quasi-clique concept to widen the common substructures.

3.4.3 Product graph construction

As mentioned in Chapter 2, a product graph is built from two input graphs.

3.4.3.1 Disadvantage of the standard product Graph

A vertex in the product graph exists, if two vertices share the same node label in the input graphs. In the case of non-relational graphs, i.e. a node label may occur many time as for the physicochemical properties of the protein binding site, the number of possible combination and therefore nodes in the product graph is exponential to the number of identical nodes in the input graphs. Figure 3.5 shows an example of all possible node combinations between graph G1 nodes and graph G2 nodes.

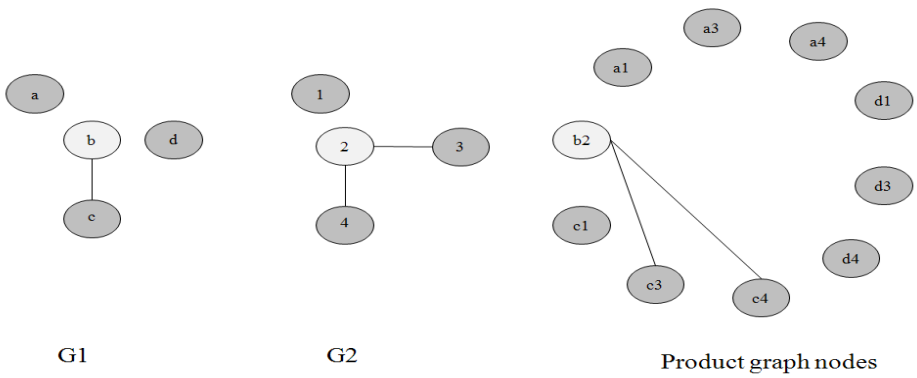


Figure 3.5: Product graph nodes construction

An edge is represented if it exists in both input graphs, or does not in the first graph neither in the second graph. With the great number of product graph nodes that have been already constructed, this method of putting edges will produce false positive matches and the construction of trivial common substructures.

3.4.3.2 A modified construction of the product graph

To reduce the size of the product graph and therefore the spatial and runtime complexity, we propose a modified manner to built the product graph. This is very useful, especially when we handle graphs of large size. The idea is to share the product graph into connected components of smaller size.

Product node construction

Nodes in the product graph are constructed as in the original manner of building the product graph. It means, if a node in the first graph shares the same node label with a node from a second graph, a product graph node is than built.

Product edge construction

To put all the edges between already existing product nodes, there are three steps.

a) Conservation of the structure step

To avoid the waste of similar substructures between input graphs, we put an edge between two product graph nodes, if an edge exists between two nodes in the first graph as well as between their matcher (according to the product node) in the second graph.

b) Pruning step

Prune product graph nodes that are not linked to any other product graph node. Our motivation through this pruning step is to reduce non trivial substructure that are not biologically significant as well as false positive matches.

Non-trivial substructures reduction The pruning of disconnected nodes that we proposed at this step, will help to reduce the number of non-trivial common substructure found. Since the maximum substructure will represent our measure of similarity it should be biologically significant.

Example 3.4. *We consider two graphs G1 an G2, and perform the construction of their product graph in the standard manner. As shown in Figure 3.6, if we consider gamma equal to one we will have as maximum cliques: {1,2,4}, {1,2,5} and {2,3,4}. For instance, {1,2,5} is not biologically significant because the single node 5 is disconnected from 1 and 2. We notice that this maximum substructure is obtained after adding edges that do not exist in both graphs G1 and G2. Pruning this node, will avoid the obtention of this common substructure.*

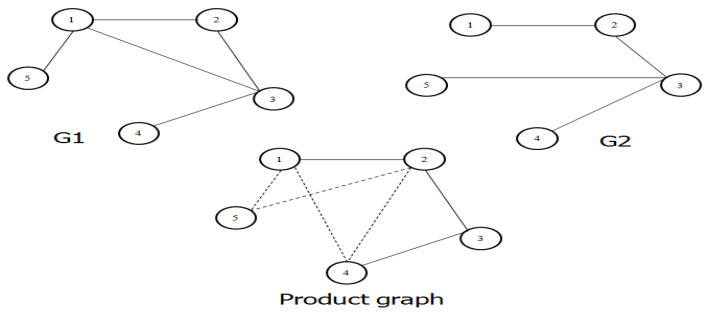


Figure 3.6: triviality of γ-quasi-cliques

False positive matches reduction A false positive match is a match between nodes that share the same node label just by chance and have different adjacency. To reduce some of these false-positive nodes, and by same way reduce the search space and runtime complexity of our method, we prune those nodes that are not connected to any other node.

Example 3.5. *Given two graphs G1 and G2 and their not fully constructed product graph P. Figure 3.7 shows that node A and node E share the same node color though they have different environment (B,G different from F,D). A mapping of A onto e (as well as E onto a) is incorrect and should not be taken into consideration to mine quasi-cliques.*

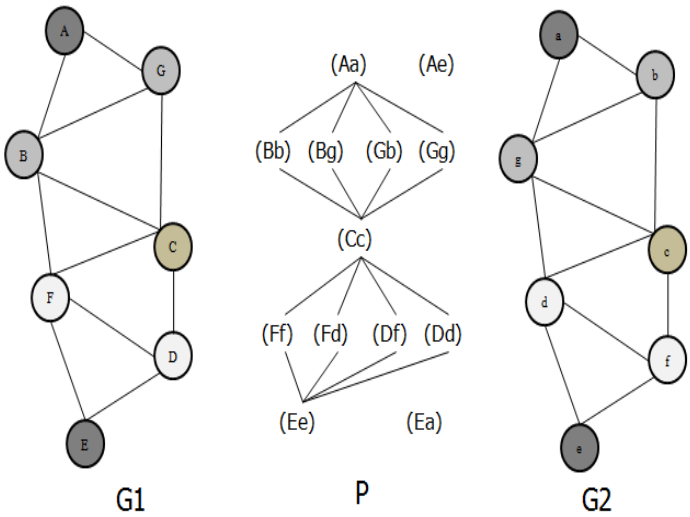


Figure 3.7: Product graph showing false-positive matches

c) Maximization of the common substructure size step

As for the standard product graph, we put an edge between two product graph nodes if an edge does not exist in both input graphs. This is done in order to maximize the size of the common substructure.

Example 3.6. *We consider two graphs G1 and G2, Figure 3.8(A) depicts their product graph without this maximization step, if we take gamma equal two one then the resulting maximum cliques are of size two: {A,B}, {B,C}, {C,D}. In Figure 3.8(B), we notice that after this maximization step we obtain {A,B,D} and {B,C,D} as maximal cliques which are of larger size (three).*

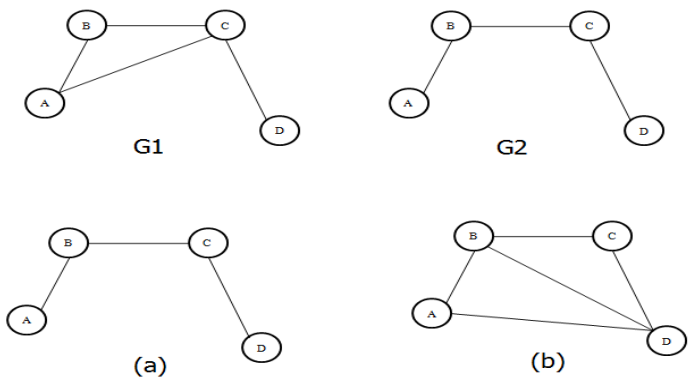


Figure 3.8: Maximization of the common substructure size

By this way, we have more meaningful connected components of smaller size, on which we will detect maximal quasi-cliques.

3.4.4 Similarity measure

The measure of similarity we have used take into consideration:

- The size of the common substructure
- The minimum number of nodes between the input graphs
- the maximum number of nodes between the input graphs

Graphs will be represented by n-by-n d(G1,G2) table, where d(G1,G2) is the measured difference or dissimilarity between two graphs G1 and G2.

Notice that the fact that our measure do not verify all metric properties is not a disadvantage in our work. In fact, it was proved that metric is an interesting property but not necessary to achieve good classification results.

3.4.4.1 Ordered Weighted Average (OWA) operator

The operator OWA (Ordered Weighted Averaging Operator) is initially proposed by (Yager, 1988) in the multi criterion aggregation framework and now it represents an important class of the operators of aggregation. These operators represent mainly the two concepts:

- The decision maker can act on aggregation by imposing a more or less strict satisfaction of the whole set of criteria: one can represent from all the criteria must be satisfied to at least a criterion must be satisfied.
- It is possible to introduce a concept of weighting for criteria and thus we have a weighted OWA: a criterion is more important if it has the highest weight.

Operators OWA are one generalization of the fuzzy operators usually used (min and max) since the latter are a particular case, it is proven by adjusting the vector of weights of an operator OWA.

3.4.4.2 Similarity measure

Two objects A and B are equals if at the same time A is a subset of B and B is a subset of A

$$A=B \leftrightarrow A\subseteq B \wedge B\subseteq A$$

To measure the similarity between two substructures A and B, we have to assess these inclusions presented before. We denote by α and β respectively the inclusion $A\subseteq B$ and $B\subseteq A$. As A and B share a common substructure identified through the quasi-clique detection step, the size of this common substructure (number of nodes denoted |QC|) will allow the assessment of these inclusions. Thus, α and β will be equal to:

- α: $\frac{|QC|}{|A|}$
- β: $\frac{|QC|}{|B|}$

The decision whatever the size of the largest or smallest structure (of A and B) should be taken into consideration, i.e. α or β, to measure the similarity is not objective. Thus, we have choose the similarity measure based on the Ordered Weighted Average fuzzy operators

to profit from the possible different weighting that offer the OWA. Hence, the measure of similarity is defined as:

$$\text{Similarity}=\text{OWA}(\alpha,\beta) \tag{3.1}$$

which can be written as:

$$\text{Similarity}=\varphi\ \min(\alpha,\beta)+(1\text{-}\varphi)\ \max(\alpha,\beta) \tag{3.2}$$

3.4.5 Graph-based similarity using QCD steps

In this section, we present the different steps to measure the similarity between a pair of graphs $G_1(V_1,E_1)$ and $G_2(V_2,E_2)$

1. The product graph P(V,E) construction

   - for each $v_1 \in V_1$ and for each $v_2 \in V_2$ compare l(v_1) and l(v_2), insert a node (v_1,v_2) into V if the labels are equal.
   - for each (v_1,v_2)$\in$ V and for each ($v_1^{'}$,$v_2^{'}$)$\in$ V compare the edge weights separating v_1,$v_1^{'}$ and v_2,$v_2^{'}$. If their difference is less or equal to a threshold ϵ then insert an edge between the nodes (v_1,v_2) and ($v_1^{'}$,$v_2^{'}$).
   - Prune nodes in the product graph that are not linked to any other nodes and thus share nodes into components.
   - for each component C(V,E) and for each $v = v_1$,$v_2 \in$ V and for each ($v^{'}v_1^{'}$,$v_2^{'}$)$\in$ V, if an edge does not exist between v_1 and $v_1^{'}$ and between v_2 and $v_2^{'}$ then insert an edge between the nodes (v_1,v_2) and ($v_1^{'}$,$v_2^{'}$).

Example 3.7. *Let us illustrate our method by a simple example. Let be two graphs representing two binding sites denoted: G1, G2. G1 is composed of 5 nodes and G2 is composed of 6 nodes.*
The nodes labels symbolize the physicochemical properties of the binding site where A,PI,D,Ar stands respectively to Acceptor, PI-PI interaction, Donor and Aromatic. The edges express the distances between these properties, these weights are represented in Table **??** *and Table* **??** *in Angström Å. Since distances inferior to 11Å are judged enough to exhibit the interactions between the pseudocenters, a distance superior to 11 Å is represented with '-' in the following tables and not illustrated in the graph representation as shown in Figure 3.9.*
Numbers nearer each node in the figure, are used to ease the comprehension of the example, hence they should not be understand as an ordering of the node.

Table 3.1: G1 adjacency matrix

G1					
	PI	A	A	PI	D
PI	0	1.22	3.53	-	9.41
A	1.22	0	2.98	-	9.75
A	3.53	2.98	0	9.39	9.07
PI	-	-	9.39	0	6.88
D	9.41	9.54	9.07	6.88	0

Table 3.2: G2 adjacency matrix

G2						
-	D	A	PI	A	PI	Al
D	0	9.70	6.70	9.00		8.79
A	9.70	0	1.27	3.10		7.25
PI	6.70	1.27	0	9.50	-	-
A	9.00	3.10	9.50	0	1.27	9.88
PI	-	-	-	1.27	0	6.85
Al	8.79	7.25	-	9.88	6.85	0

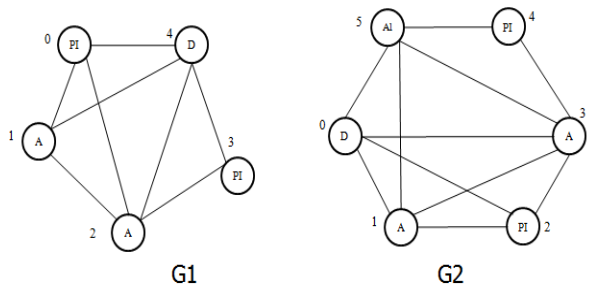


Figure 3.9: Graph representing protein binding sites

Our aim is to extract a common substructure from these two graphs. We will use it to measure the distance between them. To achieve this purpose, we start by producing a product graph, such a node exists if two nodes from G1 and G2 share the same labels. We obtain 9 nodes as shown in Figure 3.10(a). Each vertex is represented with a composed number such that the number at the left and at the right refers respectively to the number of the vertex in G1 and in G2.

From the adjacency matrices, we put an edges between product nodes if an edge exists in both graphs with a difference no more than a threshold ϵ allowed. In this example, we take ϵ equal to 0.2. Figure3.10(b) illustrate the resulting product graph.

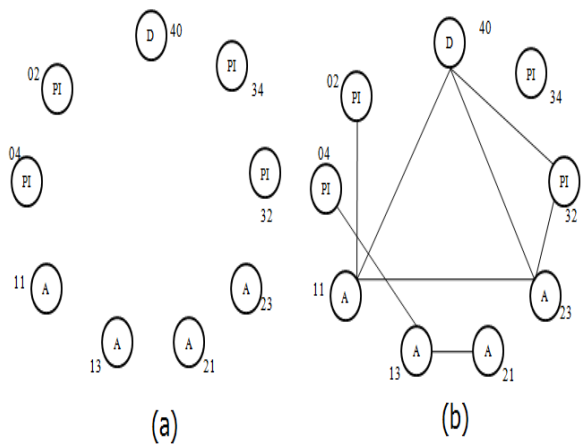


Figure 3.10: Product graph

The drawback of product graphs are their big size which make the task of detecting cliques on them hard. We propose to reduce its size by sharing it into connected components. By this way, disconnected nodes will be pruned. We start from the following boolean adjacency matrix of the graph on Figure 3.10(b) represented in Table 3.2. where a cell contains true (T) if its corresponding nodes are adjacent and false (F), otherwise.

Table 3.3: Edge matrix

	0	1	2	3	4	5	6	7	8
0	-	F	T	F	F	F	F	F	F
1	F	-	F	T	T	F	F	F	F
2	T	F	-	F	F	T	F	F	T
3	F	T	F	-	T	F	F	F	F
4	F	T	F	T	-	F	F	F	F
5	F	F	T	F	F	-	T	F	T
6	F	F	F	F	F	T	-	F	T
7	F	F	F	F	F	F	F	-	F
8	F	F	T	F	F	T	F	F	-

We obtain two components {0,2,5,6,7,8} and {1,3,4} as shown in Figure3.11. Since node 7 is not connected to any other node, it have been pruned.

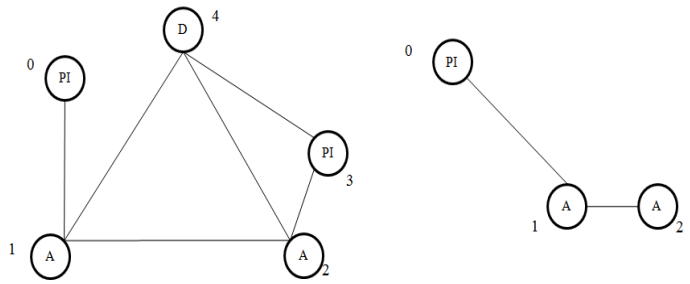


Figure 3.11: Connected components

The next step, is not add an edge in each component, if it does not exists in both graph. In this Example, we do not have edges to add, so our components will not change. From this step, we consider each component as an independent product graph on which we want to detect γ-quasi-cliques to measure the similarity between the pair of input graphs.

2. Quasi-cliques detection on the components C(V,E)

   We use pruning techniques proposed in (Liu & Wong, 2008) to reduce the search space for eventual quasi-cliques and thus significantly reduce the runtime. In our work, to mine quasi-cliques, it seems appropriate for us to use their proposed techniques. In the following, we present the progression of this quasi-clique detection step.

   - Explore the search space in a depth first order. Start with an empty set X.
   - Given X, compute *Cand_exts(X)*. However, we in our case we should not tolerate that a quasi clique contains two (or more) product nodes that share the same node in the first or second graph. Thus, to avoid such cases, we should remove nodes referring to the same node in the input graphs from the candidate extension of a vertex set, and thus presenting all possible combinations as Cand_ext(X).
   - Find the covering vertex of X and define the set of vertices that are covered by u. This latter set will be removed from *Cand_exts(X)*.
   - If the cardinality of the sum of X with *Cand_exts(X)* is less the minimum size then stop the search
   - If the union of X with *Cand_exts(X)* form a quasi-clique , then output X$\cup$*Cand_exts(X)* and skips its subsets.
   - According to the relationship between γ and the graph diameter k, prune vertices that are not in $N_k^G(v)$.
   - Compute the Lower and Upper bound that can be added to form a γ-quasi-clique and use them to prune critical vertices.
   - Compare each new γ-quasi-clique QC found with the set of γ-quasi-cliques S, if it does not have any superset in S then output QC.

Example 3.8. *Let us continue with Example 4.2. In component1, we start with an empty set, X. We notice that, node 3 and 0 refer to same node (node2) in graph G2. Hence, the candidate extensions of X should not have node 3 and node 0 together in the same subset. The resulting possible Cand_ext(X) are {0,1,2,4} and {1,2,3,4}.*

We first start with X={∅} and Cand_ext{0,1,2,4}. In this Example, we fix γ to 0.5 and the minimum size to 2. We first seek for cover vertices to prune non maximal quasi-cliques. Only the vertices in Cand_ext(X)-$C_X(u)$ where u is the cover vertex is used to extend X.

- *Cover vertex search: we search for the vertex that maximize:*
  Cand_ext(X)$\cap N_G(u) \cap (\bigcap_{v \in X \wedge (u,v) \notin E} N_G(v))$
  - *u=0: {0,1,2,4} ∩ {1} ∩ ∅*
    = {1} → size=1
  - *u=1: {0,1,2,4} ∩ {0,2,4} ∩ ∅*
    = {0,2,4} → size=3
  - *u=2: {0,1,2,4} ∩ {1,3,4} ∩ ∅*
    = {1,2} → size=2
  - *u=4: {0,1,2,4} ∩ {1,2,3} ∩ ∅*
    = {1,2} → size=2

  The cover vertex is 1 and $C_X(1)$={0,2,4} → Cand_ext={1, 0,2,4}.
  Cand_ext(X)- $C_X(1)$ = [1].

- *Check if X ∪ Cand_ext(x) inferior to the minimum size that we have fixed for the quasi-clique. 4≥2 so we continue.*
- *Look ahead technique: check if X ∪ Cand_ext(x) {1,0,2,4} form a 0.5-quasi-clique.*
  - *deg(1)=3*
  - *deg(0)=1*
  - *deg(2)=2*
  - *deg(4)=2*

  The minimum degree found is 1. To obtain a 0.5-quasi-clique, we should have at least a degree of 0.5(n-1). Here, n is equal to 4 and then to form a 0.5-quasi-clique the minimum should be no less than 0.5*(4-1)=1.5. This requirement is not satisfied so {1,0,2,4} is not a 0.5-quasi-clique.*

- *Let us consider a vertex set Y={X}∩ the ith vertex in Cand_ext(X)- $C_X(1)$.*
  On Cand_ext(X) we will remove this vertex added to Y.
  Candidates extension of Y denoted Candy are computed as
  Cand_ext(X) ∩ N_k^G of the last element added to Y, where k is computed as the upper bound of the diameter of the graph. For a 0.5 quasi-clique, k is equal to 2.
  Y={1}, Candy= {{0,2,4} ∩ {0,2,4}}={0,2,4}.

- *Next, we compute the upper bound, U and the lower bound, L of the number of vertices that should be added to the current vertex set to form a 0.5-quasi-clique.*

- *U: The maximal number of vertices that can be added to Y to form a 0.5-quasi-clique should be no larger than the minimum degree of the vertices in Y divided by γ+1-|Y| is denoted U_{min}. U_{min}: $\frac{2}{0.5}$+1-1=1.*

  U=max{t|$\sum_{v \in Y}$ $indeg^Y(v)$+
  $\sum_{1 \leq i \leq t}$ $indeg^X(v_i) \geq$ |Y|(γ *(|Y|+t-1)*
  and 1$\geq$ t$\geq$ U_{min}} if such t exists
  Otherwise, U is equal to zero. Cand_ext(Y),v_i, are sorted according to their $indeg^Y$ value. $indeg^Y$ represents the number of vertices in Y that are adjacent to the vertex v.
  - *indeg(0)=1*
  - *indeg(2)=1*
  - *indeg(4)=1*

  Let us compute U:
  - *$\sum_{v \in Y}$ $indeg^Y(v)$ = 0*
  - *for t=1, $\sum_{1 \leq i \leq t}$ $indeg^Y(v_i)$=1,*
    |Y|(γ *(|Y|+t-1)= 1*(0.5+1-1)=0.5, 1$\geq$ 0.5, U=1.*
- *L: The number of vertices need to be added to Y to increase the degree of u (u $\in$ Y) in order to form a 0.5-quasi-clique is denoted L_Y^{min}. It is equal to the minimum t such that the minimum indeg degree, $Indeg_{min}$ of the vertices in Y plus t is greater or equal to (γ*(|Y|+t-1)). $Indeg_{min}$=1, for t=0, 0+0$\geq$(0.5*(1+0-1)), L_{min}= 0*
  L=min{t|$\sum_{v \in Y}$ $indeg^Y(v)$+
  $\sum_{1 \leq i \leq t}$ $indeg^Y(v_i) \geq$ |Y|(γ *(|Y|+t-1)*
  and $L_{min} \geq$ t$\geq$ n} if such t exists. Otherwise, L is equal to |Cand_ext(Y)|+1.
  Let is compute L:
  - *$\sum_{v \in Y}$ $indeg^Y(v)$ = 0*
  - *for t=1, $\sum_{1 \leq i \leq t}$ $indeg^Y(v_i)$=1,*
    |Y|(γ *(|Y|+t-1)= 1*(0.5+1-1)=0.5, 1$\geq$ 0.5, L=1.*
- *We will use these two bounds to remove iteratively the invalid candidate extensions of Y.*
  - *Check if Y contains a critical vertex. These nodes satisfy that the sum of their degrees, indeg and extdeg, is equal to [γ*(|Y|+L-1)]. The only one node in Y, u=1 do not have this property, since 2 $\neq$ 0.5*(1+1-1).*
  - *Check if Y contains invalid vertices, i.e. v $\in$ Y, that satisfy, according to their degree (indeg and extdeg), one of these conditions:*
    - *$indeg^Y(v)$+$extdeg^Y(v)$l[γ*(|Y|+$extdeg^Y(v)$-1)]*
    - *$indeg^Y(v)$+U l[γ*(|Y|+U-1)]*
    - *$indeg^Y(v)$+$extdeg^Y(v)$l[γ*(|Y|+L-1)]*

    On v=1:
    - *0+2$\geq$(0.5*(1+2-1))*
    - *0+1$\geq$(0.5*(1+1-1))*
    - *0+2$\geq$(0.5*(1+1-1))*

    Y does not contains such invalid vertex.

- *Check the extensibility of the vertices in Y, i.e. $v \in$ Cand_ext(Y):*
  - $indeg^Y(v)$+$extdeg^Y(v)$l[γ*(|Y|+$extdeg^Y(v)$)]
  - $indeg^Y(v)$+U-1 l[γ*(|Y|+U)]
  - $indeg^Y(v)$+$extdeg^Y(v)$l[γ*(|Y|+L-1)]

  Candy=0,2,4.

  - *v=0: 1+0≥(0.5*(1+0)), 1+1-1≥(0.5*(1+1)), 1+0≥(0.5*(1+1-1)*
  - *v=2: 1+1≥(0.5*(1+1)), 1+1-1≥(0.5*(1+1)), 1+1≥(0.5*(1+1-1)*
  - *v=4: 1+1≥(0.5*(1+1)), 1+1-1≥(0.5*(1+1)), 1+1≥(0.5*(1+1-1)*

  All the vertices in Candy, will be kept.

We repeat the same process with X={Y=1} and Cand_ext={Candy={0,2,4}}.

- *Cover vertices:*
  - *u=0: {0,2,4}∩{1}∩∅=∅: size=0*
  - *u=2: {0,2,4}∩{1,3,4}∩∅=4: size=1*
  - *u=4: {0,2,4}∩{1,2,3}∩∅=2: size=1*

  2 and 4 are of size one, we choose node 2 as a cover vertex. Thus, Cand_ext={0,2, 4}
- *First of all, node 1 will be extended with node 0. Y=1,0 and Cand_ext={2,4}, thus Candy={{4,2}∩{1,2,4}}={2,4} .*
  - *The upper bound computation:*
    deg_{min}=1+0=1, U_{min}=$\frac{1}{0.5}$+1-2=-0.75, no possible value for t, since t should be positive and more than U_{min} then U=0.
  - *The lower bound computation:*
    $indeg_{min}$=1, L_{min}=0 $\sum_{v \in Y}$ $indeg^Y(v)$=1+1=2. For t=0, we have 2+1 ≥ 2(0.5*(2+0-1)) then L=0.*
  - *Critical vertices:*
    - *u=0: 1+0 ≠(0.5*(2+0-1)*
    - *u=1: 1+2 ≠(0.5*(2+0-1)*
  - *Check invalid vertices in Y.*
    - *u=0: 1+0≥(0.5*(2+0-1)), 1+0≥(0.5*(2+0-1)), 1≥(0.5*(2-1))*
    - *u=1: 1+2≥(0.5*(2+2-1)), 1+0≥(0.5*(2+0-1)), 3≥(0.5*(2-1))*
  - *Check the extensibility of X with vertices in Candy*
    - *u=2: 1+1 ≥ 0.5*(2+1), 1+0-1 ł0.5*(2+0-1) → 2 is a bad extension for {0,1}*

  * *u=4: 1+1 ≥ 0.5*(2+1), 1+0-1 ł0.5*(2+0-1) → 4 is a bad extension for {0,1}*

  Candy=Candy-{2,4} Candy={∅}, {0,1} not a quasi-clique.

- *Then node 1, will be extended with node 2. Y={1,2}, Cand_ext={0,4} thus Candy={{0,4}∩{1,3,4}}*
  - *The upper bound computation:*
    deg_{min}=min3,2=2, U_{min}=$\frac{2}{0.5}$+1-2=-2, no possible value for t, since t should be positive and more than U_{min}. U=0.
  - *The lower bound computation:*
    $indeg_{min}$=min1,2=1, L_{min}=0, no possible value for t, L=2+1=3.
  - *Critical vertices:*
    * *u=1: 1+2 ≠(0.5*(3+2-1))*
    * *u=2: 1+1 =(0.5*(3+2-1))→ critical vertex.*
      N^G(2)={1,3,4}. Candy={4}∩{1,3,4}={4}.
      Y={1,2}∩Candy={1,2,4} and Candy={∅}
  - *check if Y is a 0.5-quasi-clique: deg_{min}= 2, to form a 0.5-quasi-clique, each vertex should have at least {0.5*((n=3)-1)=1} degree. |Y|≥ to the fixed minimum size of the quasi-clique, it forms a quasi-clique and non of its supersets is a 0.5-quasi-clique: Our input is 1,2,4.*

After that , we repeat the same process with the second combination we have found of the candidate extension of X: X=∅ and Cand_ext{1,2,3,4}.
1,2,3,4 is of size 4 which is greater than 2, then we check if {1,2,3,4} is a 0.5-quasi-clique. In fact the minimum degree of these vertices is 2 which is greater or equal to 0.5(4-1). We this lookahead pruning technique, we stop the search and output {1,2,3,4} as a maximal quasi-clique for component 1. Once, we have our result for the first component, we move to the second one (Figure 3.11)(b). We start with X={∅} and Cand_ext={0,1,2}.*
{0,1,2} is of size 3 which is greater than 2, then we check if {0,1,2} is a 0.5-quasi-clique. In fact the minimum degree of these vertices is 1 which is greater or equal to 0.5(3-1). We this lookahead pruning technique, we stop the search and output {0,1,2} as a maximal quasi-clique for the component 2.*

3. Maximal quasi-cliques extraction

   - Construct a prefix tree on the set of quasi-cliques found previously
   - For each leaf, mark its subsets as non-maximal
   - Output leafs that are not marked as non-maximal

Example 3.9. *Given the graphs presented in Example 4.2. We compare the size of the maximal clique found for each component, and keep the maximum one. Thus our prefix tree will contain {1,2,4} and {1,2,3,4}. Since {1,2,4} is a subset of {1,2,3,4} it will be marked as non-maximal. Thus, our final common substructure for the first component is {1,2,3,4} with a size of 4 (see Figure 3.12).*

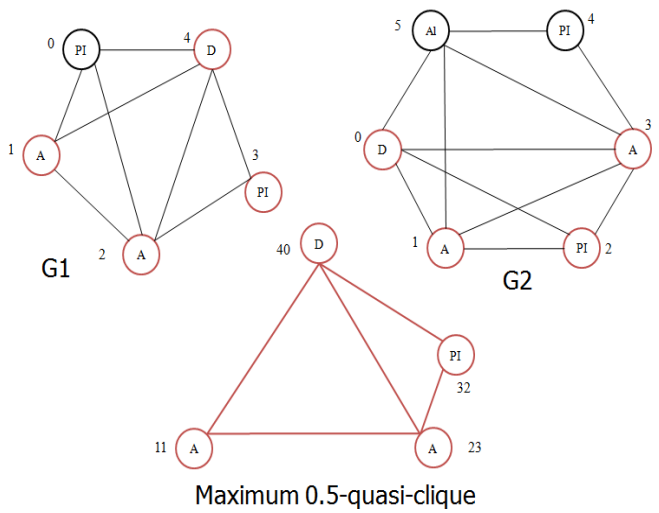


Figure 3.12: The maximum common substructure between G1 and G2

4. Similarity measure

   - Compute the similarity between G_1 and G_2 according to the Equation 3.2.

   Example 3.10. *Given the results of last examples, if we took φ equal to 0.2 then G_1 and G_2 will be similar according to:*

$$\begin{array}{rrcll} & Sim(G_1,G_2) & = & \varphi \cdot min(\alpha,\beta) & + & (1-\varphi)\cdot max(\alpha,\beta) \\ \leftrightarrow & Sim(G_1,G_2) & = & 0.2 \cdot min(\frac{4}{5},\frac{4}{6}) & + & (1-0.2)\cdot max(\frac{4}{5},\frac{4}{6}) \\ \leftrightarrow & Sim(G_1,G_2) & = & 0.773 & & \end{array}$$

Remark: We notice through these examples that if we have taken γ equal to 1, only the first maximal quasi-clique found on the first component, i.e. {1,2,4} will be detected thus G_1 and G_2 will be separated by a distance of 0.58 for the same value of φ. It proves the efficiency of our method to detect larger common substructure and thus to improve classification.

3.5 Conclusion

In this chapter, we have presented our method for the assessment of the similarity between graphs representing protein binding pockets structure. Our method is a graph-based similarity method which is based on the quasi-clique detection as a measure of similarity.

Our method is a kind of relaxation of the clique measure. It is used in order to have larger common substructures between the input graphs. We have exposed a modified manner to construct the product graph to reduce false-positive matches and trivial common substructures and a similarity measure allowing to assign structures to appropriate class family. Then, we have detailed the different steps allowing the graph comparison.

In the next chapter, we will present the implementation for checking the performance of our measure comparing to the standard clique similarity. Then, we will show different results obtained from simulations and that have been performed on real databases.

Chapter 4

Implementation and experimentation

4.1 Introduction

Implementing our graph-based similarity based on quasi-cliques is important since it allows us to have an idea concerning the feasibility of our proposed method. In this chapter, we present the implementation of our method to measure the similarity between graphs representing protein binding pockets. To this end, we have developed all programs in Java language with ECLIPSE 3.4.1.

Once the different programs are implemented, for checking the feasibility of our approach and judging its performances, we have performed several tests and simulations on several real databases obtained from CavBase (Schmitt et al., 2002), a database system for the fully automated detection and extraction of protein binding pockets from experimentally determined protein structures (available through the PDB database (Berman et al., 2000)).

Different results will be presented and analyzed in order to assess the efficiency of our measure.

This chapter is composed of two parts. First, Section 4.1 deals with the implementation of our graph similarity measure derived from quasi-clique detection where the major variables and programs are detailed. The principal algorithms are also exposed. Section 4.2 is interested to the simulation phase. Then, results over data representing protein active sites cavities are presented with an analysis of them. Note that the objective of this simulation is the feasibility and the efficiency of our similarity measure.

4.2 Implementation

4.2.1 Framework

As mentioned above, in order to ensure the implementation of similarity measure, we have developed programs in java language with ECLIPSE 3.4.1. Obviously, we have implemented our comparison method that we have detailed in the previous chapter.
These programs have as inputs:

1. A data set in the ".rlbcoor" format from Cavbase (Schmitt et al., 2002), a database system for the automated detection, extraction, and storing of protein cavities (hypothetical binding pockets) from experimentally determined protein structures (available through the PDB). In Cavbase, graphs are used as a first approximation to describe binding pockets (see Appendix A for further details).
2. The cutoff value for edge distances.
3. The value of the tolerance threshold for edge length differences.
4. The value of the minimum degree threshold allowing the detection of the quasi-cliques.
5. The minimum size required to form a quasi-clique.
6. The number of neighbors for the K-NN classifier.

The outputs of our programs are:

1. The set of maximal quasi-cliques on the product graph for a pair of graphs
2. The similarity between a pair of graphs.
3. The accuracy of the similarity measure given by the Percent of Correct Classification (PCC) (a widely used criterion for measuring the accuracy of classifiers). This criterion will be detailed in the following sections.

4.2.2 Principle variables

The major variables that we have used in our programs to implement the graph-based similarity that are based on quasi-clique detection are the following:

- *path_file:* the path of the inputfile of graphs.
- *parameter:* an object that is composed of:
  - *.γ:* a user specified parameter representing the minimum degree threshold, its value is between 0 and 1.
  - *.δ:* the value of the tolerance threshold for edge length differences, by default equal to 0.2.
  - *.ϵ:* the cutoff value for edge distances, by default equal 11Å.
  - *.φ:* the weight for the similarity measure criteria, its value is between 0 and 1.
  - *.min_size:* the minimum size required to form a quasi-clique.
  - *.k:* the number of neighbors that should be taken into consideration for the K-NN classifier.
- *diam:* the upper bound of the diameter of the graph.
- *graph:* an object that is composed of:
  - *.path:* a string indicating the path file of the graph.
  - *.name:* the name of the molecule.

  - *.numberofnodes:* the number of nodes (physicochemical properties) of an input graph
  - *.nodes:* a
    with the nodes labels
  - *.distance:* a matrix of doubles containing the distances between different nodes.

- *Merge:* a vector of strings that contains all the files name in the dataset.
- *File:* a string that indicates the name of a given structure.
- *product_graph_node:* a structure composed of a $node_i$ from a first graph G_i and a $node_j$ from a second graph G_j
- *product_graph:* an object that is composed of:
  - *.edges:* a boolean matrix representing the adjacency matrix.
  - *.nodes:* a table of product_graph_node.
  - *.productnodes:* a list of product_graph_node.
- *components*: a list of component.
- *neighbors:* a vector containing the set of the neighbors of a product_graph_node.
- *X:* a vertex set in the search space.
- C_u*:* a list of vertices that are covered by a cover vertex u.
- *allCand:* a vector of vector of the candidates extension of X.
- *Cand_ext(X):* the valid set of candidates extension of X.
- *lqc:* a vector of vector of subsets in the search space.
- *qc:* a boolean variable that indicate if current vertex set form a quasi-clique.
- *Cqc:* a list of nodes that forms a quasi-clique.
- *U:* an integer that indicate the maximum number of nodes that must be added to form a quasi-clique.
- *L:* an integer that indicate the minimum number of nodes that must be added to form a quasi-clique.
- *indeg:* the number of vertices in a vertex set X that are adjacent to a specified vertex.
- *exdeg:* the number of vertices in the list cand_exts(X) that are adjacent to a specified vertex.
- *Max:* the size of the maximal quasi-clique found.
- *SimilMatrix:* a matrix that contains all pairwises comparisons
- *class:* the functional class family of the protein binding site.
- *sim:* a number that quantify the similarity between a pair of graphs.
- *PCC:* the percent of correct classification of the class of option tree.

4.2.3 Main programs

In this subsection, we will present the major programs that we have developed to construct our software. We will regroup these programs according to the task they are used for.

Graph creation

- **Parse:** creates the graph distance matrix and the list of nodes labels and removes edges with a distance greater than the cutoff value. This procedure has as parameter the path of the file to parse.
- **Create_graph:** creates a new graph. It has as parameter the path file and delta the cut-off value for edges. It calls the procedure Parse.

Product Graph construction

- **Product_graph_node:** builds a node of the product graph as a merging of two nodes sharing the same label from the input graphs.
- **Product_graph:** creates a product graph from a pair of graphs and share it into disconnected components.
- **AddEdge:** adds edges into components if an edge does not exist in both input graphs.
- **Merging:** merges in the same vector product graph' nodes that share the same neighborhood.

Quasi-clique detection

- **findQC:** A principal recursive function that checks if a set of vertices according to its candidate extension subset contains a quasi-clique and output the set of quasi-clique. It calls these following functions:
  - **Cand_ext:** returns the valid list of candidates that can extend a subset X.
  - **Indeg:** computes the number of vertices in X that are adjacent to a given vertex.
  - **Extdeg:** computes the number of vertices in Cand_ext of X that are adjacent to a given vertex.
  - **QuasiClique:** checks if a set of vertices form a γ-quasi-clique.
  - **Cover:** identifies the cover vertex that maximize the result given by the function cover.
    - **cover:** returns the set of vertices that are covered by a given vertex.
  - **Critical:** returns a list of critical vertices in a given vertex set X.
  - **K_Neighbors:** seeks for the k neighbors of a given vertex where k is the upper bound of the diameter of the graph. It calls PruneDiam.
    - **PruneDiam:** computes the upper bound of the diameter of the graph to be a quasi-clique.
  - **U:** computes the number of vertices that can be added to form a γ-quasi-clique.
  - **L:** computes the number of vertices that can be added to form a γ-quasi-clique.

Measuring Similarity

- **Similar:** computes the distance a pair of graphs according to Equation 3.2.
- **Simil:** calls the function Similar for each pair of graphs and save these comparisons into a matrix denoted SimilMatrix.
- **EvalSim:** evaluates our measure by computing its PCC: the percentage of correctly classified graphs according to the functional family they belong to.

4.2.4 Graph-based similarity derived from the quasi-clique detection algorithms

In this section, we will present the major algorithms relative to our method to measure the similarity between a pair of structure cooresponding to each pahse, namely the creation graph, the product-graph construction, the quasi-clique discovery and the similarity measure algorithms.

Creation graph phase

Algorithm: Create_graph
input: path_file, δ
output: graph

```
begin
(* Parsing a given input file for graph creation *)
Parse(path)
(* Removing edges with distance greater cutoff value *)
trimDistance(delta);
(* Creating List of nodes labels *)
createNodes();
(* Creating a graph g out of a given file *)
g←Graph(numberOfNodes, distances, nodes, coords, path);
end
```

Product graph construction phase

Algorithm: Product_graph
input: graph G1, graph G2, ϵ
output: components

```
begin

(* Creating product graph nodes *)
for i←0 to numberNodes(G1) do
  for j←0 to numberNodes(G2) do
    (* Testing if two nodes from the graph G1 and G2 share the same node label*)
    if  G1.nodes[i] = G2.nodes[j] then
      (* Creating a product graph node, denoted node *)
      node.node1←i;
      node.node2←j;
      (* Addind this node to productNodes *)
      productNodes←Add(node);
    end if
  end for
end for

(* Create product graph edges *)

(* First Step: structure conservation*)
for  i←0 to size(productNodes) do
  for j←i to size(productNodes) do
    (* Checking if the distance between two nodes in G1 and their corresponding nodes in G2 are
    separated by the same distance or with a difference of a threshold ϵ *)
    if  (((G1.distances[nodes[i].node1][nodes[j].node1] =
        G2.distances[nodes[i].node2][nodes[j].node2])
        and (G1.distances[nodes[i].node1][nodes[j].node1] != 0))
        or ((Math.abs(G1.distances[nodes[i].node1][nodes[j].node1]) -
        (G2.distances[nodes[i].node2][nodes[j].node2])≤ϵ))) then
      (* Putting true in the corresponding cell of the adjacency matrix *)
      edges[i][j]←true;
      edges[j][i]←true;
    else
      (* Put false if they are not neighbors*)
      edges[i][j]←false;
      edges[j][i]←false;
```

```
      end if
    end for
  end for

  (* Second Step: creating components *)

  (* Computing the number of neighbors for each node i and put them into a vector vect *)
  for i←0 to length(adjacency matrix) do
    (* Checking if it is a disconnected node *)
    if size(neighbors)≥1 then
      vect←vect + neighbors(i);
    end if
  end for
  (* Merging the neighbors in a common vector if they share nodes (i.e. neighbors) *)
  vect←Merging(vect);
  (* Components creation for each set of nodes according to their partition in vect *)
  for i←0 to size(vect) do
    component← Createcomp(vect.get(i));
  end for
  components← Add(component);

  (* Third Step: size maximization *)

  (* Adding edges if they do not exist in both graphs *)
  for i←0 to size(components) do
    AddEdge(components(i));
  end for
  end.
```

Quasi-clique detection phase

Algorithm: FindQC
Input: X, Cand_ext(X), γ, min_size
Output: Cqc

```
(* Finding the cover vertex of X and output the set of vertices that are covered by it*)
cover←Cover(X,Cand_ext,adjacent);
(* Putting the set of the vertices in cover after all vertices of Cand_ext(X) *)
for i←0 to size(cover) do
  j←Cand_ext.indexOf(cover.get(i));
  Cand_ext.add(cover.get(i));
  Cand_ext.remove(cover.get(i));
end for
(* Initializing the current vertex set to false, i.e. not a quasi-clique *)
qc←false;
for i←0 to size(Cand_ext) - size(cover) do
  (* Checking if the subset composed of X and Cand_ext(X) satisfies the min_size fixed size *)
  if |X| + |Cand_exts(X)| < min_size then
    findQC←false;
    node.node1←i;
  end if
```

```
(* Lookahead technique: Checking if the subset composed of X and Cand_ext(X) form a quasi-clique *)
F←X∩Cand_exts(X);
if QuasiClique(F,adjacent,parameter.gamma) then
   findQC←true;
   Cqc.Add(F);
end if
(* Cheking if the min_size requirement is satisfied and the superset is not a quasi-clique extend the current subset X with the i^th value in Cand_ext *)
Y← X ∩ Cand_ext(i)
Cand_ext← Cand_ext - Cand_e;xt(i);
(* Diameter pruning: identifing the set of vertices N^k(Cand_ext(i)) *)
K_N←K_Neighbors(Cand_ext(i),F, adjacent,γ);
(* Computing the valid set of candidates extension of Y denoted Candy *)
for  x←0 to size(Cand_ext) do
   for y←0 to size(K_N) do
      (* Keeping only nodes that are in K_N *)
      if  Cand_ext.get(x) = K_N.get(y) then
         Candy.add(Cand_ext.get(X));
      end if
   end for
end for
(* Upper bound and lower bound pruning computation *)
U←U(Candy,Y,allnodes,adjacent,γ,PN);
L←L(Candy,Y,allnodes,adjacent,γ,PN);
repeat
   (* Checking if there is a critical vertex in Y *)
   critical←Critical(Y,allnodes,adjacent,L,γ,PN);
   if critical != -1 then
      V←neighbors(critical,adjacent);
      for x←0 to size(V) do
         for y←0 to size(Candy) do
            (* Keeping only nodes that are in V *)
            if Candy.get(y).equals(V.get(x)) then
               Y.add(Candy.get(y));
               Candy.remove(Candy.indexOf(Candy.get(y)));
            end if
         end for
      end for
      U←U(Candy,Y,allnodes,adjacent,γ,PN);
      L←L(Candy,Y,allnodes,adjacent,γ,PN) ;
   end if
   (* Checking if there is invalid candidates in the current subset*)
   Z←InvalidY(Y,U,L,γ);
   (* If Z non empty no need for further extensions *)
   if size(Z) = 0 then
      clear(Candy);
   end if
   (* Checking if there is invalid candidates in Candy *)
   Z←InvalidC(Candy,U,L,γ);
   Candy.removeAll(Z);
until L > U and size(Z) = 0 and size(Candy)=0
if L≤U and size(Candy) != 0 and |Y|+|Candy|≥min_size then
```

```
68:       (* Checking if there are a superset of it that form a quasi-clique *)
69:       supqc←findQC(Y,Candy,γ,min_size);
70:       qc←qc or supqc;
71:       (* Checking if Y and non of its superset is a quasi-clique *)
72:       if |Y|≥ min_size and QuasiClique(Y,adjacent,γ) and supqc = false then
73:         findQC←true;
74:         Cqc.Add(Y);
75:       end if
76:     end if
77:   end for
78:   end.
```

Algorithm: Solution

input: graph G1, graph G2, parameter
output: Max

```
1:  begin
2:  (* Creating the product graph of G1, G2 *)
3:  productGraph←ProductGraph(g1,g2,parameter.epsilon);
4:  (* Generating one candidate extension Cand *)
5:  Cand←candidate(X,productGraph.productNodes,allNodes);
6:  (* Generating all candidates extension according to Cand *)
7:  allCand←Cand_Ext(X,productGraph.productNodes,allNodes,Cand);
8:  (* Detecting maximal quasi-cliques according to each possible Cand_ext in allCand*)
9:  for  y←0 to size(allCand) do
10:    findQC(X, allCand.get(y), allNodes, productGraph.edges,
       productGraph.productNodes, parameter, quasicliques);
11:    qc←quasicliques;
12:    lqc.add(qc);
13:    (* Detecting maximal quasi-cliques for this combination of Cand_ext *)
14:    Max←Maximal(lqc.get(y));
15: end for
16: end.
```

Similarity measure

Algorithm: Simil

```
1:  input: Merge, File, parameter, δ
2:  output: Sim
3:  begin
4:  (* Initialization *)
5:  ϕ←parameter.phi;
6:  sim←-1;
7:  α←0;
8:  β←0;
9:  min←-1;
10: queryFile←Merge;
11: sampleFile←File;
12: graphs[0]←parseFile(queryFile, delta);
13: graphs[1]←parseFile(sampleFile, delta);
14: (* Computing the size of the maximum quasi-clique founc between graph[0] and graph[1] *)
```

```
SizeQCMax←Solution(graphs[0], graphs[1], parameter);
(* Checking if there is a quasi-clique between the input graphs *)
if sizeQCMax != -1 then
   (* Computing α and β *)
   α←SizeQCMax / graphs[0].numberOfNodes;
   β←SizeQCMax / graphs[1].numberOfNodes;
   (* Assessing the similarity between the graphs according to the similarity measure defined in
   Chapter 3*)
   if αlβ then
      min←α;
   else
      min←β;
   end if
   if min = α then
      max←β;
   else
      max←α;
   end if
   sim←ϕ * min + (1 - ϕ) * max;
end if
end.
```

Algorithm: EvalSim

```
input: Merge, parameter
output: PCC
begin
(* Making all pairwises comparisons *)
SimilMatrix←classif(Merge, parameter);
(* Sorting each row in SimilMatrix *)
SimilMatrix←sortM(SimilMatrix);
(* For each row in SimilMatrix picking the k most high value for the K-Nearest Neighbor classifier*)
SimilMatrix←Kmatrix(SimilMatrix);
for i←0 to length(SimilMatrix) do
   (* for each structure (row i) predicting its class according to the vote of the K-Nearest Neighbor
   classifier*)
   class←Vote(SimilMatrix[i])
   if strcmp(true-class, class) = 1 then
      Correct←Correct+1;
   end if
end for
(* Computing PCC *)
PCC← Correct / size(Merge) * 100;
end.
```

4.3 Experimentation

4.3.1 Experimental Setup

The implementation of our algorithm will be useful in the simulation phase. We have performed several tests and simulations were build upon Cavbase database (Schmitt et al., 2002). Different results carried out from these simulations will be presented and analyzed in order

to evaluate our proposed method.

4.3.2 Evaluation criteria

4.3.2.1 Percent of Correct classification: PCC

The PCC measure the performance of our algorithm. It represents the percent of the correct classification. It is given by:

$$\text{PCC} = \frac{\text{number of well functional classified structures}}{\text{total number of classified structures}} \times 100 \tag{4.1}$$

The PCC is computed by comparing for each instance between its real class (the family to which it belongs (a 2-class classification problem: ATP[1] or NADH [2])) and the class given by the k -nearest neighbor classification according to our similarity measure. Hence, the number of well classified graph structures represents the number of instances for which the class obtained by the major vote ok the k most similar structures is the same as their real functional family.

When the obtained PCC is equal to 100%, it means that this measure is an excellent measure, whereas a null measure has a PCC equal to 0%.

4.3.3 Validation criteria

In our method, we used the *one-leave out cross validation* with *k-nearest-neighbor classifier*. It consists on the following steps:

1. Pick one graph structure, say G_i.
2. Search for the k structures in the whole data set that are maximally similar to G_i by looking at the similarity matrix (the k maximal values in the i-th row of the matrix, excluding position i); of course, we are not allowed to pick G_i itself.
3. Make a majority vote, which means that we determine the class which is most frequent among the k nearest neighbors; call it c. It will represent the estimated class for G_i.
4. Check whether class G_i is effectively equal to c, i.e., whether we have predicted the right class.

We repeat these steps for all structures in the data set and determine the percentage of correct predictions.

4.3.4 Simulation on real data sets

4.3.4.1 Description of the data sets

Our work is build upon CavBase(Schmitt et al., 2002), a database of cavities on protein surfaces that present a chemical characterization of cavities and can detect unexpected similarities amongst protein cavities (e.g. active sites) that share little or no sequence homology.

[1]ATP: Adenosine triphosphate
[2]NADH: Nicotinamide adenine dinucleotide

We have used a data set containing 54 instances belonging to the ATP [3] and NADH [4] family in the .rlbcoor format. distributed in the following way:

Table 4.1: Description of the database

Functional family	Number of instances
ATP	26
NADH	27

See Appendix B for further details about CavBase and ATP-NADH dataset.

4.3.4.2 Experimental results

In our simulations, we were interested by the impact of varying the value of γ, the minimum degree of vertices to form a γ-quasi-clique, as well as the value of φ in Equation 3.2, the parameter of the similarity measure that we have defined (see Section 3.4.2), on the PCC.

Note that the experimentations were performed using an Intel Core Duo CPU (2.4 GHz) with 2 GB of RAM running Windows Vista.

Our simulations will be performed for two cases, according to the value of γ, cliques, and quasi-clique similarity measure.

1. Clique similarity measure : The first case tests the efficiency of the standard clique detection to measure the similarity between a pair of input graphs when we apply our measure.

2. Quasi-clique similarity measure : The second case tests the efficiency of our graph-based similarity measure method according to the different values of γ.

Results of the clique measure

Note that the standard clique measure, is equivalent to the quasi-clique measure for γ equal to one. Table 4.2 summarizes the different accuracy relative to ATP-NADH dataset for γ equal to one when applying the 1-nearest-neighbor classifier vote. We compare the PCC for two standard cases where respectively only the size of the smaller graph (φ=1) and the larger graph (φ=0) is used for to assess the similarity between graphs with our similarity measure.

[3] Adenosine triphosphate
[4] Nicotinamide adenine dinucleotide

Table 4.2: The impact of the variation ϕ values on the PCC

φ	k=1
1	75.92
0	74.02
0.1	75.92
0.2	77.77
0.3	81.48
0.4	83.33
0.5	81.48
0.6	81.48
0.7	79.62
0.8	79.62
0.9	79.62

From this table, we can conclude that our measure method for γ equal to one has good results. There is an improvement of the PCC. Indeed, it goes from 74.02% for (φ=0) and also from 75.92% for (φ=1) to 83.33% for (φ=0.4).

Results of the quasi-clique measure

Figure 4.1 shows the correlation between an increasing value of γ and the improvement of the PCC.In this figure, we represent the mean PCC according to the different values that can take φ. The results prove that the relaxation of the clique concept improves graph comparisons and thus classification task. In fact, the PCC increases from 79.11% for γ equal to one it means for cliques to 88,88% for γ equal to 0.5.

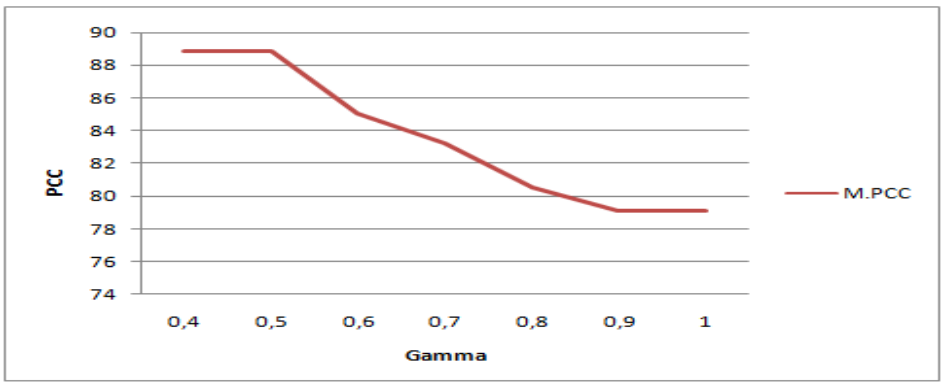


Figure 4.1: Mean PCC according to φ values vs gamma values

Since there is an increase on the PCC, it means that the variation of gamma is correlated with the size of the maximum substructure found. Table **??** presents this relationship between two structures of the dataset: 2eww.1.rlbcoor and 2r6x.1.rlbcoor. Notice that their corresponding graphs have respectively 19 and 21 nodes.

γ	mcs	similarity
1	6	0.3097
0.9	6	0.3097
0.8	8	0.4133
0.7	9	0.4646
0.6	9	0.4646
0.5	9	0.4646
0.4	10	0.5162
0.3	11	0.5679

From the tendency of the curve in Figure 4.2, we conclude that the size of the common substructure is positively correlated with increasing values of γ.

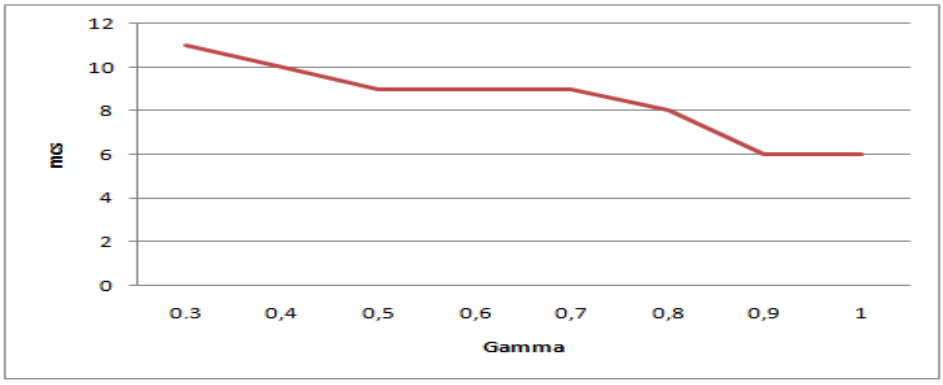


Figure 4.2: Size maximum quasi-clique vs gamma value

We have to mention that thought discovering quasi-cliques increases the size of the common subgraph, it is more time expensive than cliques. In fact, there are a correlation between the value that take gamma and the runtime. Figure 4.3, shows the difference in runtime between γ=1 and γ=0.5 in average according to the number of graphs.

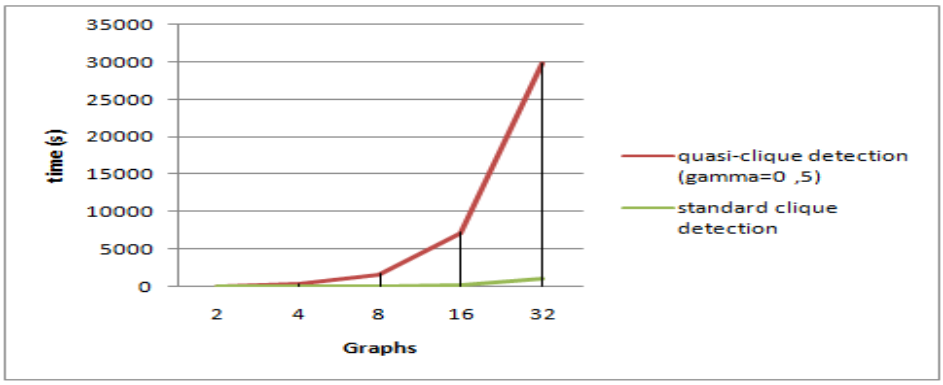


Figure 4.3: Runtimes

Thu, we should take into consideration this runtime constraint when choosing a value for γ to make a compromise between quality and time consumption.

4.4 Conclusion

In this chapter, the major variables , the main programs and the principal algorithms that we have used in order to implement our similarity measure on graphs, are detailed. Experimental results, provided in this chapter, show that our new approach (described in Chapter 3) gives better results than the standard version of this method using the special case where γ is equal to one. These results are interesting since they show that according to the value that γ takes, the flexible nature of the protein structure will not affects trivially structures similarity. The obtained results from simulations that have been performed on real databases are mentioned and have shown encouraging results.

Conclusion and future work

In this report, we have defined a new graph similarity measure, based on the quasi-clique concept for the structural analysis of protein binding sites. Our method allows the identification of region of correspondence between two protein active sites' cavities and its use to measure the similarity between them.

Since the graph representation of protein structure avoid the loss of information, the use of graph matching techniques is appropriate to compare two graphs and assess their similarity according to their common substructure. Our approach aims to optimize the size of this common substructure in a geometrically and biologically meaningful manner.

Clique-detection algorithms were proved to be efficient to implement the maximum common subgraph problem and thus represent a measure of similarity between them. Our idea, is to use a relaxed concept of cliques named γ-quasi-cliques where the value of the term γ relax the notion of clique by fixing a minimum degree of the vertices instead of the standard (n-1) used with cliques. Even if γ-quasi-cliques are a generalization of cliques when γ equal to one, existing algorithms for mining cliques can not be used to detect quasi-cliques. Indeed, the downward closure property does not hold for quasi-cliques.

Due to their flexible nature on the 3D-level, related proteins can take different conformations. A graph representation of a distorted binding site will be represented by missing edges, hence a lower degree of the vertices. While clique detection algorithms will penalize these misrepresentations and output two related proteins as far similar, the use of γ-quasi-cliques takes into consideration, according to the value of γ, such distortions when identifying the common substructure.

We have defined a graph-based measure that take into account the size of the common substructure between the two input graphs, identified through the quasi-clique detection phase on their product graph, according to the size of the compared graphs to assess the similarity between two protein binding sites.

We have performed simulations on real databases in order to evaluate the performance of our graph-based similarity measure. Using the PCC as an evaluation criterion, the results of the experiments on data sets show the efficiency of our approach. The similarity between graphs based on quasi-cliques discovery can outperform the standard clique detection approach since there is a correlation between the size of the common substructure and a decreasing value of γ.

An interesting future work is to make a multiple alignment using pairwises graph comparisons. In fact, discovering maximum quasi-clique on a pair of graphs can be seen as a local alignment between them. If a make multiple comparisons always according to a pivot graph and keep a node that is conserved in almost comparisons, we obtain a multiple alignment. The resulting functional consensus graph will serve to better classify proteins.

Appendix A

Alignment Algorithms

A.1 Introduction

The way to align sequences differ from the way to align graphs. In fact, the order of amino-acids in sequences do not hold when dealing with graphs. Besides, matching a pair of object differ from the problem of matching multiple objects. In this appendix, we will present major algorithms used in bioinformatics to align either sequences or graphs.

A.2 Sequence alignment

A.2.1 Pairwise sequence alignment algorithms

The alignment task can be done with either *exact* algorithms or *heuristics*.

A.2.1.1 Exact algorithms

- Dot Matrix Methods
  It provides a complete picture of homology of protein sequences. It illustrates *graphically* regions of similarity between two sequences. Hence, to compare two proteins, a matrix is built by putting one sequence across the top and one sequence down the left side. For each position (i,j) in the grid, if column i and row j are equal then a dot is represented at this position.

  Example A.1. *Consider two sequences: kasply and yaspfe. The following matrix implicitly produces a family of alignments for individual sequence regions, in this example only 'asp' is found. This method is simple, however it can become time-consuming on a large scale.*

Table A.1: Dot matrix

	k	a	s	p	l	y
y						·
a		·				
s			·			
p				·		
f						
e						

- Dynamic Programming In this section we present, two well known algorithms that use dynamic programming to obtain a pairwise sequence alignment: the Needleman-Wunsch algorithm and the Smith-Waterman algorithm.

  - The Needleman-Wunsch algorithm (Needleman & Wunsch, 1970) uses a matrix called F matrix whose $(i, j)^t h$ entry is often denoted F_{ij}. Given two sequences 1 and 2, there is one column for each character in sequence 1, and one row for each character in sequence 2. Gap penalty is noted by d. Alignments will be represented by pathways in this matrix. F_{ij} will be assigned to be the optimal score. It is calculated recursively as:

$$F_{ij}= \max(F_{i-1,j+1} + \mathrm{S}(B_i,A_j),F_{i,j-1} + \mathrm{d},\ F_{i-1,j} + \mathrm{d}) \tag{A.1}$$

Example A.2. *Let us continue with the two sequences used in the preceding example: kasply and yaspfe. The matrix is initialized according to gaps penalties :*

Table A.2: F-Matrix initialization

		k	a	s	p	l	y
	0	-1	-2	-3	-4	-5	-6
y	-1						
a	-2						
s	-3						
p	-4						
f	-5						
e	-6						

Table A.3: F-Matrix

		k	a	s	p	l	y
	0	-1	-2	-3	-4	-5	-6
y	-1	-2	-3	-4	-5	-6	2
a	-2	-2	2	1	0	-1	1
s	-3	-2	1	6	5	4	3
p	-4	-3	0	5	13	12	11
f	-5	-4	-1	4	12	13	15
e	-6	-4	0	3	11	12	14

Table A.4: Optimal alignment on an F-Matrix

		k	a	s	p	l	y
	0	-1	-2	-3	-4	-5	-6
y	-1	$\overbrace{-2}$	-3	-4	-5	-6	2
a	-2	-2	$\overbrace{2}$	1	0	-1	1
s	-3	-2	1	$\overbrace{6}$	5	4	3
p	-4	-3	0	5	$\overbrace{13}$	$\overbrace{12}$	11
f	-5	-4	-1	4	12	13	$\overbrace{15}$
e	-6	-4	0	3	11	12	$\overbrace{14}$

Then, according to a similarity matrix (e.g BLOSUM), F_{ij} is computed. For example : for i=1, j=1 and d=-1 and from Equation A.1

= Max((0+-2),(-1+-1),(-1+-1))= -2

The resulting matrix is presented in Matrix A.3.

Through the F matrix the alignment with the maximal score can be easily identified. In this example it will obtains a score of 14. To find this alignment, start with the cell $F_{lenght_1,lenght_2}$, here $F_{6,6}$ and after that move in the matrix according to the way that we obtain through it the current cell score. It means if the value of the current cell is equal to:

* *$F_{i-1,j-1}$ + s(i,j) then move according to the diagonal*
* *$F_{i-1,j}$ − d then turn left*
* *$F_{i,j-1}$+ d then go up.*

The optimal alignment is shown in Matrix A.4. Thus, the optimal alignment found is the following and it has a spatial and temporal complexity of $\mathcal{O}$(MN) where M= lenght1 and N=lenght2

k a s p - l y
y a s p f e -

– The Smith-Waterman algorithm (Smith & Waterman, 1981): It is a variation of the Needleman-Wunsch algorithm which tries to find the best match between parts of the sequences. Starting by initializing the matrix with zeros, the F_{ij} will be modified as follow:

$$F_{ij}= \max(0,\ F_{i-1,j+1} + \mathrm{S}(B_i,A_j), F_{i,j-1} + \mathrm{d},\ F_{i-1,j} + \mathrm{d}) \tag{A.2}$$

The stopping criteria is a cell scoring 0 encountered. These modifications will encourage local alignments.

Example A.3. *Let us apply the Smith-waterman algorithm on the two sequences treated in the last example. The resulting matrix will contains:*

Table A.5: Smith-Waterman F-Matrix

		k	a	s	p	l	y
	0	0	0	0	0	0	0
y	0	0	0	0	0	0	2
a	0	0	2	1	0	0	1
s	0	0	1	6	5	4	3
p	0	0	0	5	13	12	11
f	0	0	0	4	12	13	15
e	0	0	0	0	11	12	14

The alignment found:

k	a	s	p	-	l	y
y	a	s	p	f	e	-

We notice that is the same alignment that already found by the Needleman-Wunsch algorithm, which means that it is not necessary that the Smith-Waterman algorithm gives a different alignment. There is a difference only if the two sequences are dissimilar. The complexity is still $\mathcal{O}(MN)$.

A.2.1.2 Heuristics

With heuristics, there is not a guarantee to find the best alignment between two sequences. Tools detailed here, FASTA[1] (Lipman & Pearson, 1985) and BLAST[2] (Altschul et al., 1990), are based on word methods heuristics. In fact, positions of words in the two sequences have to be compared. These positions are subtracted to obtain an offset. The same offset among distinct letters indicates a region of alignment.

- FASTA
  It is a global sequence alignment algorithm that tries to find paths of regional similarity, rather than trying to find the best alignment between two sequences. It allows mismatches and thus alignments can contain gaps, though they will be penalized. For proteins FASTA search for 1-tup (i.e. subsequences of length equal to 1) between two sequences and scores it with PAM matrix.

  Example A.4. *To align two proteins sequences : kasply and yaspfe*

Position	1	2	3	4	5	6	7	8	9	10	11
*Protein*1	k	a	s	p	l	y	$-$	$-$	$-$	$-$	$-$
*Protein*2	$-$	$-$	$-$	$-$	$-$	y	a	s	p	f	e

[1]Fast Algorithm
[2]Basic Local Alignemnt Search Tool

Table A.6: FASTA: comparison between amino-acids positions

Amino Acid	position in protein 1	position in protein 2	pos1-pos2
y	6	6	0
a	2	7	-5
e	-	11	
k	1	-	
p	4	9	-5
f	-	10	
s	3	8	-5
l	5	-	

$$\begin{array}{lcccccc} Apossiblealignment: & Protein1 & k & a & s & p & l & y \\ & & & \| & \| & \| & & \\ & Protein2 & y & a & s & p & f & e \end{array}$$

FASTA saves the 10 best local regions, regardless of whether they are on the same of different diagonals that was seen on a dot matrix plot. They will be scored with the scoring matrix. If there are several initial regions with scores greater than the cutoff value, we will see whether these regions can be joined to form an approximate alignment with gaps.

- BLAST

The idea behind it is that good alignments contain short lengths of exact matches. Blast is faster then FASTA. The algorithm works as follow :
Given a query sequence, it will divided in subsequences, i.e. words of length of three letters. These words will be aligned with other words called neighborhood words in the database, that score at least T using a substitution matrix.

Example A.5. *Given this query sequence: EFPQGL*
The result after its division to three letter words is the following:

EFP

FPQ

PQG

QGL

A substitution matrix (BLOSUM for example) is used to score these pairs with words in the database. Suppose that for the word 'PQG' and with these words in the database, scores are:

PQG=18 PEG=15 PRG=14 PKG=14 PQA=12 PQN=12

If a threshold T is fixed to 14 then the neighbor words will be only PQG, PEG, PRG, PKG. This process is done for all subsequences.

These words will be extended to find a locally optimal alignment. A high scoring pair (HSP) will be extended. A cutoff S is used to keep only extended sequences that have a score at least S. Thus, local alignments without gaps are obtained.

A.2.2 Multiple sequence alignment algorithms

There are many algorithms to build a multiple alignment. According to NotreDame (Notredame, 2002), they can be divided into exact algorithms using dynamic programming and heuristics like progressive or iterative or consistency based algorithms.

A.2.2.1 Exact Algorithms

These algorithms use dynamic programming (Gupta et al., 1995). They are a generalization of the Needleman-Wunsch algorithm to N-dimensions. However, their use is computationally expensive. In fact, time grows as product of sequence lengths. Therefore, it becomes intractable for more than a few sequences or large sequences.

Example A.6. *Having these three sequences:*

- *KASPY*
- *ASPF*
- *FY*

The goal is to find the alignment that maximizes the score function. The alignment will be modeled through an optimal path found on the dynamic programming matrix as shown in Figure A.1.

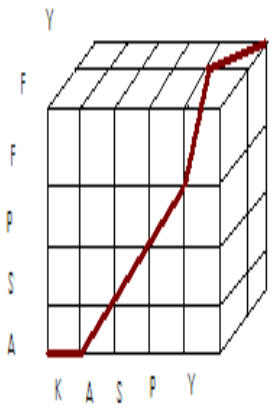


Figure A.1: Optimal path through dynamic programming matrix returns the multiple alignment

The optimal alignment of this 3 sequences is:

K	A	S	P	-	Y
-	A	S	P	F	-
-	-	-	-	F	Y

For the multiple sequence alignment problem using various criteria of optimality. It has been shown that this problem is NP-Hard (Wang & Jiang, 1994). That is why many methods use approximate algorithms that leads to a trade-off between optimality and speed.

A.2.2.2 Progressive alignments

Algorithms using progressive alignments are heuristic, so finding the optimal alignment is not guaranteed. In fact, they form a multiple alignment of two sequences. Then they align the consensus of this multiple alignment with another sequence producing an expanded multiple alignment. This process is repeated until all sequences are aligned. The order in which sequences are aligned is important. Given n sequences, $S_1 \ldots S_n$, it is likely that you will obtain a different multiple alignment for each permutation of $S_1 \ldots S_n$ (Notredame, 2002). For this reason, alignment algorithms such as ClustalW (Tompson et al., 1994) use what is called a guide tree which is a a phylogenetic tree based on distance calculated from pairwise similarity score for all pairs to determine the order for the alignment. The algorithm starts by comparing all sequences pairwise and then performs a cluster analysis. It generates a hierarchy, so it matches closely-related sequences first using a guide tree and then aligns this alignment with next similar alignment or sequence.

A.2.2.3 Iterative methods

By iterative, a multiple re-building is meant. These re-buildings are constructed in order to try to find the best alignment among all proteins sequence. Iterative methods start with a non-optimal alignment and try iteratively to optimize it. Iterative algorithms, are divided into stochastic and non-stochastic ones.

Stochastic algorithms: A stochastic algorithm is one that incorporates some degree of randomness into the solution. Though they can be slow when they are applied to align a big number of proteins, they produce a high quality alignments. Simulated annealing algorithms as used for MSASA[3] (Kim et al., 1994) and where alignment is randomly modified as only acceptable alignment kept for further process until convergence or genetic algorithms as used for SAGA[4](Notredame & Higgins, 1996) which are evaluated using an objective function are good examples of stochastic algorithms.

Non-stochastic algorithms: They are deterministic algorithms. The mistake in alignment is rectified by iteration. Realignments are performed until no further improvement are found in next iterations. The quality of the alignment is based on whether or not the iterations have reached convergence. As non-stochastic iterative algorithms we can mention algorithms used in AMPS[5](Barton & Sternberg, 1987) as well as PRRP (Gotoh, 1996).

A.2.2.4 Consistency based algorithms

The optimal alignment in a consistency based algorithm is the one that agrees best with all the possible pairwise alignments. This is an NP complete problem, so all consistency based algorithms are heuristics. These algorithms are based on any method that is able to do a pair-wise alignment of two sequences. This method is used to produce a library of alignments for each pair. This library of alignments is then used to ensure consistency in the resulting multiple sequence alignment.

Consistency can be ensured as the alignment pairs are used to create a position specific substitution matrix. Alignment pairs have been created using both local and global alignment algorithms. The advantage of this substitution matrix over the other substitution matrices (BLOSUM, PAM) is that it is based on the current sequences, and the substitution scores are a measure of how likely it is that a base will appear in a given position. Once this new position specific substitution matrix is created the multiple sequence alignment is created using a progressive alignment similar to that used in ClustalW (Notredame, 2002).

A.3 Structural alignment

A.3.1 Pairwise structural alignment algorithms

Protein structure is closely to function than protein sequence. In fact, structural motifs may predict similar biological function. To align two proteins, each of them is considered as a geometric object in the plane and will be used to make a superposition of one protein onto another. It means, a transformation (rotation, translation, scaling) of these two sets of points is sought. Two proteins of the same length are optimally superimposed, if the RMSD is minimal. (Hendrickson, 1979).

- Geometric-intermolecular: Algorithms might do this by minimizing the RMSD in superimposed alpha-carbon positions

[3]Multiple Sequence Alignment by Simulated Annealing
[4]Sequence Alignment using Genetic Algorithm
[5]Alignment of Multiple Protein Sequences

- Geometric-intramolecular: Algorithms minimize the difference between aligned contact maps or distance matrices.
- Non-Geometric: Algorithms align structural properties, such as secondary structure. This is usually done with dynamics programming.

Algorithms can be divided to those using monte carlo simulation (e.g.Distance matrix-based ALIgnment (DALI)(Holm & Sander, 1995), heuristics (e.g. Secondary Structure Alignment Program (SSAP)(Orengo & Taylor, 1996)), those that use dynamic programming (e.g. Combinatorial Extension of the maximum path (CE) method (Shindyalov & Bourne, 1998)) and probabilistic ones. In the following, we present them:

- DALI: It is an intramolecular geometric structural alignment algorithm. It generates a distance matrix D for each protein such as D_{ij} is equal to the distance between alpha carbon i and alpha carbon j. When two proteins' distance matrices share the same or similar features in approximately the same positions, they can be said to have similar folds with similar-length loops connecting their secondary structure elements.

- SSAP: It uses dynamic programming. It is based on atom-to-atom vectors in structure space. Instead of the alpha carbons typically used in structural alignment, SSAP constructs its vectors from the beta carbons. SSAP works by first constructing a series of inter-residue distance vectors between each residue and its nearest non-contiguous neighbors on each protein. A series of matrices are then constructed containing the vector differences between neighbors for each pair of residues for which vectors were constructed. Dynamic programming applied to each resulting matrix determines a series of optimal local alignments which are then summed into a matrix to which dynamic programming is applied again to determine the overall structural alignment.

- CE: it is bases on heuristics and also uses dynamic programming.CE aligns two polypeptide chains A and B using characteristics of their local geometry as defined by vectors between C alpha positions. Matches are termed as aligned fragment pairs (AFPs). An AFP is defined as a continuous segment of A aligned against a continuous segment of B of the same size and without gaps. The idea is to start by selecting some initial AFP and then build an alignment path by incrementally adding AFPs, and inserting gaps between AFPs in A or in B but not in both. Heuristics are used in defining a set of optimal paths joining AFPs with gaps as needed. The path with the best RMSD is subject to dynamic programming to achieve an optimal alignment.

A.3.2 Multiple structural alignment algorithms

As for sequences, there are progressive iterative and consistency based algorithms. We present two example of algorithms using respectively a progressive and a consistency based method.

A.3.2.1 Progressive algorithm

Progressive alignments methods are also used to align several structures as used in (Taylor et al., 1994). This method is a fusion of the fast structure comparison program SSAP and

the multiple sequence alignment program MULTAL. Structures are progressively combined, producing intermediate consensus structures that are compared directly to each other and all remaining single structures.

A.3.2.2 Consistency based algorithm

It tries to find conflict-free subsets of alignments from a pre-computed all-to-all pairwise alignment library (PAL) (Chen & Crippen, 2006). This library must reduce conflicts as possible to avoid generating erroneous alignments from the beginning and then increases chances to find optimal alignment. This is done in the Iterative Refinement of Induced Structural alignment algorithm (IRIS) (Chen & Crippen, 2006) by refining the pairwise library to reduce the number of conflicts before any consistency-based assembly.

A.4 Conclusion

In this appendix, we have presented some algorithms in the literature that are used to align a pair or more than two either sequences or structures.

Appendix B

Cavbase database

B.1 Introduction

To benefit from all the biological information from the research area, it should be collected into databases where data can be grouped together based on biologically meaningful similarities. These databases are seen as resources for biological research. Biological databases are libraries of life sciences information, collected from scientific experiments, published literature, high throughput experiment technology, and computational analysis. Information contained in biological databases includes gene or protein function and structure as well as similarities of biological sequences and structures. Actually, the extraction of data and querying biological databases must accord the interest of a given researcher. Available databases are divided according to their content in generalized databases (or data banks) and specialized databases.

B.2 CavBase

It is developed by S. Schmitt (Univ. Marburg) (Schmitt et al., 2002). It automatically extract binding sites from protein structure determined experimentally available through the PDB database (Berman et al., 2000).

PDB is the protein Data Bank on proteins of Research Collaboratory for Structural Bioinformatics. It is a world collection of data on the 3-dimensional structure of biological macromolecules: proteins, primarily, and nucleic acids.

CavBase stores the binding site composition amino-acids, the attributed pseudocenters(3D-descriptors) and the cavity surface. The pseudo center concept is used as a simple descriptor to code molecular recognition as shown in Figure B.1.

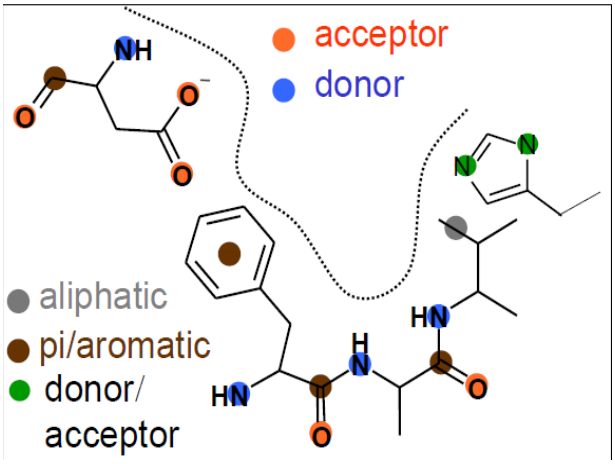


Figure B.1: pseudocenters assignment

The Cavbase approach is a cavity detection based on the LIGSITE (Hendlich et al., 1997) program. The Database currently contains 113,718 hypothetical binding pockets that have been extracted from 23,780 publicly available protein structures using the LIGSITE algorithm.

LIGSITE is a new program for the automatic and time-efficient detection of pockets on the surface of proteins that may act as binding sites for small molecule ligands. Pockets are identified with a series of simple operations on a cubic grid. Using a set of receptor-ligand complexes we show that LIGSITE is able to identify the binding sites of small molecule ligands with high precision.

The CavBase cavity database is closely linked to, and may be used alongside **Relibase+** which is a web-based application and all its functionality is accessible via a web browser such as Netscape or Internet Explorer.

B.3 NADH-ATP dataset

We used a two-class data set of 54 protein binding pockets : Adenosine-5'-triphosphate (ATP) and nicotine amide dinucleotide (NADH). Each family share a certain prominent cofactor. We used Cavbase to retrieve all known ATP and NADH binding pockets that were co-crystallized with the respective ligand. Subsequently, we reduced the set to one cavity per protein, thus representing the enzymes by a single binding pocket.

B.3.1 Adenosine triphosphate- ATP

ATP is the molecule of universal energy. It is the power behind the life. The ATP is a molecule made up of adenine, ribose and a phosphate molecule as shown in Figure B.2. In all forms of life, it is the principal conveyor of energy. It is estimated that the ATP produced 95% of cellular energy.

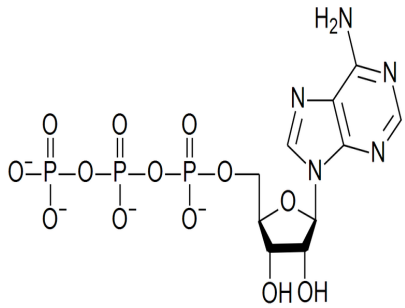


Figure B.2: Adenosine triphosphate

B.3.2 Nicotinamide adenine dinucleotide - NADH

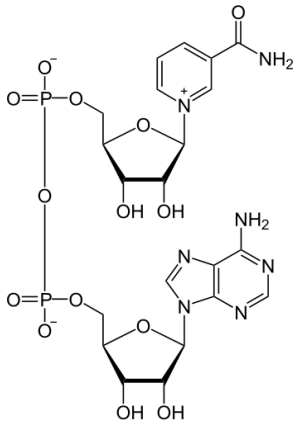


Figure B.3: Nicotinamide adenine dinucleotide

NADH is a coenzyme found in all living cells. The compound is a dinucleotide, since it consists as shown if Figure B.3 of two nucleotides joined through their phosphate groups: with one nucleotide containing an adenine base, and the other containing nicotinamide. In metabolism, NADH is involved in redox reactions, carrying electrons from one reaction to another. It is an important cofactor or assistant who helps the enzymes in their work through the organism. In particular, it plays a vital part in the energy production of each cell. It also takes part in the production of L-dopa which the organism transforms into dopamine, an important neurotransmitter. Studies evaluated the interest of the NADH in the treatment of the jet lag, of the diseases of Alzheimer and Parkinson, of the chronic syndrome of tiredness and also as nutritional supplement for the sportsmen.

B.4 Conclusion

In this appendix, we have presented a binding sites cavities database: CavBase and we have presented some information about the ATP-NADH data set in CavBase which we have used in the simulation phase.

References

Abello, J., Resende, M. G. C., & Sudarsky, R. (2002). Massive quasi-clique detection. In *In latin american theoretical informatics (latin* (p. 598-612).

Alberts, A. (1998). *Essential cell biology: An introduction to the molecular biology of the cell* (G. Publishing, Ed.). Taylor and Francis.

Altschul, S., Gish, W., Miller, W., Myers, E., & Lipman, D. (1990). Basic local alignment search tool. *J. Mol. Biol.*, *215*(3).

Artymiuk, P., Poirrette, A., Grindley, H., Rice, D., & Willet, P. (1994). A graph-theoretic approach to the identification of three-dimensional patterns of amino acid side-chains in protein structure. *J. Mol. Biol.*, *243*, 327-344.

Attwood, T. K., & Parry-Smith, D. (1999). *Introduction to bioinformatics.* Prentice-Hall.

Barton, G., & Sternberg, M. (1987). A strategy for the rapid multiple lignment of protein sequences: Confidence levels from tertiary structure comparisons. *J Mol. Biol.*, *198*, 327337.

Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., et al. (2000). The protein data bank. *Nucleic Acids Research*, *28*, 235-242.

Bienenstock, V. d. m. C., E. (1987). A neural network for invariant pattern recognition. *Europhys Letters*, *3*, 121-126.

Branden, C., & Tooze, J. (1999). *Introduction to protein structure.* Garland publishing.

Bron, C., & Kerbosch, J. (1973). Algorithm 457 - finding all cliques of an undirected graph. *Communications of the ACM*, *16*, 575-577.

Bunke, H., & Allermann, G. (1983). Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, *1*, 245-253.

Carillo, H., & Lipman, D. (1988). The multiple sequence alignment problem in biology. *SIAM J. Appl. Math.*, *48*(5), 1073-1082.

Chen, Y., & Crippen, G. (2006, July). An iterative refinement algorithm for consistency based multiple structural alignment methods. *Bioinformatics*, *12*(17), 2087-2093.

Cohen, J. (2004). Bioinformatics-an introduction for computer scientists. *ACM Computing Surveys (CSUR)*, *36*(2), 122-158.

Crick, F. H. C. (1968). The origin of the genetic code. *Journal of Molecular Biology*, *38*(3), 367-379.

Dayhoff, M., Schwartz, R., & Orcutt, B. (1978). A model of evolutionary change in proteins. *Atlas of protein sequence and structure*, *5*, 345-358.

Dreyfus, P. (1962, jun). L'informatique. *Gestion,Paris*, 240-241.

Environmental Protection Agency. (2002). *Interim policy on genomics.* Sience Policy Council.

Erdmann, M. (2005). Protein similarity from knot theory: Geometric convolution and line weavings. *J. Comp. Biol.*, *12*(6), 609-637.

Fober, T., Hüllermeier, E., & Mernberger, M. (2008). Evolutionary construction of multiple graph alignments for the structural analysis of biomolecules. In *German conference on bioinformatics.*

Garey, M., & Johnson, D. (1979). *Computers and intractability.* Freeman and Co., New York.

Gilbert, D. R., Westhead, D. R., Nagano, N., & Thornton, J. M. (1999). Motif-based searching in tops protein topology databases. *Bioinformatics*, *5*(4), 317326.

Golan, D., Tashjian, A. H., & Armstrong, A. W. (2008). *Principles of pharmacology: The pathophysiologic basis of drug therapy.* Lippincott Williams & Wilkins.

Gotoh, O. (1996). Significant improvement in accuracy of multiple protein sequence alignments by iterative refinement as assessed by reference to structural alignments. *J Mol Biol*, *264*(4), 823-38.

Gupta, S., Kececioglu, J., & Schffer, A. (1995). Improving the practical space and time efficiency of the shortest-paths approach to sum-of-pairs multiple sequence alignment. *J. Comp. Biol.*, *2*, 459-472.

Hattori, O. Y. G. S., M., & Kanehisa, M. (2003). Development of a chemical structure comparison method for integrated analysis of chemical and genomic information in the metabolic pathways. *J. Am. Chem. Soc.*, *125*(39), 1185311865.

Hendlich, M., Rippmann, F., & Barnickel, G. (1997). Ligsite: automatic and efficient detection of potential small molecule-binding sites in proteins. *J. Mol. Graph.*.

Hendrickson, W. (1979). Transformations to optimize the superposition of similar structures. *Acta Cryst.*, *A35*, 158-163.

Henikoff, S. (1992). Amino acid substitution matrices from protein blocks. *PNAS*, *89*, 10915-10919.

Holm, L., & Sander, C. (1995). Dali: A network tool for protein structure comparison. *Trends Biochem Sci.*, *20*(11), 478-480.

Hopcroft, J., & Wong, J. (1974). Linear time algorithm for isomorphism of planar graphs (preliminary report). In *Stoc* (p. 172-184).

Hubbard, T., Lesk, A., & Tramontano, A. (1996, April). Gathering them in to the fold. *Nature Structural Biology*, *3*(4), 313.

Hüllermeier, E., Weskamp, N., Klebe, G., & Kuhn, D. (2007). Graph alignment: Fuzzy pattern mining for the structural analysis of protein active sites. In *Fuzzy systems conference.*

Karp, P. (2001). Pathway databases: A case study in computational symbolic theories. *Science*, *293*, 2040-2044.

Kim, J., Pramanik, S., & Chung, M. (1994). Multiple sequence alignment using simulated annealing. *Comput Appl Biosci*, *10*(4), 419-26.

Koike, A., & Takagi, T. (2004). Prediction of protein-protein interaction sites using support vector machines. *Protein Engineering Design and Selection*, *17*(2), 165-173.

Levenshtein, V. I. (1966). Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics-Doklady*, *10*, 707-710.

Levi, G. (1972). A note on the derivation of maximal common subgraphs of two directed or undirect graphs. *Calcolo*, *9*, 341-352.

Levin, J., Robson, B., & Garnier, J. (1986). An algorithm for secondary structure determination in proteins based on sequence similarity. *FEBS Lett.*, *205*, 303-308.

Levitt, M. (1976). A simplified representation of protein conformations for rapid simulation of protein folding. *J Mol. Biol.*, *104*, 59-107.

Lipman, D., & Pearson, W. (1985). Rapid and sensitive protein similarity searches. *Science*, *227*, 1435-1441.

Liu, G., & Wong, L. (2008). Effective pruning techniques for mining quasi-cliques. In *Ecml pkdd '08: Proceedings of the european conference on machine learning and knowledge discovery in databases - part ii* (pp. 33–49). Berlin, Heidelberg: Springer-Verlag.

Luscombe, N. M., Greenbaum, D., & Gerstein, M. (2001). *What is bioinformatics? an introduction and overview* (Tech. Rep.). Year Book of medical informatics.

Matsuda, H., Ishihara, T., & Hashimoto, A. (1999). Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theor. Comput. Sci.*, *210*(2), 305–325.

Mount, D. (2004). *Bioinformatics: Sequence and genome analysis.* CSHL Press.

Needleman, S., & Wunsch, C. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol.*, *48*(3).

Neuhaus, M., & Bunke, H. (2007). *Bridging the gap between graph edit distance and kernel machines* (Vol. 68; H. Bunke & P. Wong, Eds.). World scientific.

Notredame, C. (2002, Jan). Recent progress in multiple sequence alignment: a survey. *Pharmacogenomics*, *3*(1), 131-144.

Notredame, C., & Higgins, D. (1996). Saga: Sequence alignment by genetic algorithm. *Nucl. Acids Res.*, *24*(8), 1515-1524.

Ochagavia, M., & Wodak, S. (2004). Progressive combinatorial algorithm for multiple structural alignments: application to distantly related proteins. *Proteins*, *55*(2), 436-454.

Ohlstein, E., Ruffolo, R. J., & Elliot, J. (2000). Drug discovery in the next millennium. In *Ann rev pharmacol toxicol* (Vol. 40, p. 177-91).

Orengo, C., & Taylor, W. (1996). Ssap: sequential structure alignment program for protein structural comparison. *Methods Enzymol*, *266*, 617-635.

Parsons, R. J., Forrest, S., & Burks, C. (1995). Genetic algorithms, operators, and dna fragment assembly. *Machine learning*, 11-33.

Pei, J., Jiang, D., & Zhang, A. (2005). On mining cross-graph quasi-cliques. In *In kdd* (p. 228-238).

Raymond, J., & Willet, P. (2002). Maximum common subgraph isomorphism algorithms for the matching of chemical structures. *Journal of Computer-Aided Molecular Design*, *16*, 521-533.

Royal Academy of Engineering. (2007, Feb). *Systems biology: a vision for engeneering and medecine* (Tech. Rep.). The Academy of Medical Sciences and The Royal Academy of Engineering.

Rymond, R. (1992). Search through systematic set enumeration. In *Proc. of the international conference on principles of knowledge representation and reasoning.*

Saitou, N., & Nei, M. (1987). The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Mol. Biol. Evol.*, *4*(4), 406-425.

Schmitt, S., Kuhn, D., & Klebe, G. (2002). A new method to detect related function among proteins independent of sequence and fold homology. *J.Mol.Biol*, *323*(2), 387-406.

Searls, D. B. (1992). The linguistics of dna. *Amer.Sci.*, *80*, 579-591.

Selkow, S. (1977). The tree-to-tree editing problem. *Information Processing Letters*, *6*(6), 184-186.

Shatsky, M., Nussinov, R., & Wolfson, H. (2002). Multiprot a multiple protein structural

alignment algorithm. *Workshop on Algorithms in Bioinformatics, Springer Verlag, Lecture Notes in Computer Science*, *2452*.

Shindyalov, I., & Bourne, P. (1998). Protein structural alignment by incremental combinatorial extention (ce) of the maximun path. *Protein engineering*, *11*(9), 739-747.

Smith, T., & Waterman, M. (1981). Identification of common molecular subsequences. *J Mol Biol.*, *147*, 195-197.

Sneath, P., & Sokal, R. (1978). Numerical taxonomy. *Freeman*.

Stark, A., Sunyaev, S., & Russell, R. (2003). A model for statistical significance of local similarities in structure. *J. Mol. Biol.*, *326*, 1307-1316.

Sutcliffe, M. J., Haneef, I., Carney, D., & Blundell, T. L. (1987). Knowledge based modelling of homologous proteins, part i: Three dimensional frameworks derived from the simultaneous superposition of multiple structures. *Protein Eng.*, *1*, 377-384.

Taylor, W., Flores, T., & Orengo, C. (1994). Multiple protein structure alignment. *Protein Sci.*, *3*, 1858-1870.

Tompson, J., Higgins, D., & Gibson, T. (1994). Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. *Nucl. Acids Res.*, *22*(22), 4673-4680.

Ullman, J. (1976, January). An algorithm for subgraph isomorphism. *Journal of the ACM*, *23*(1), 31-42.

Wang, L., & Jiang, T. (1994). On the complexity of multiple sequence alignment. *J. Comput. Biol.*, *1*, 337-348.

Weskamp, N., Hullermeier, E., Kuhn, D., & Klebe, G. (2007). Multiple graph alignment for the structural analysis of protein active sites. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, *4*(2), 310-320.

Yager, R. R. (1988). On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Trans. Syst. Man Cybern.*, *18*(1), 183-190.

Zeng, Z., Wang, J., Zhou, L., & Karypis, G. (2006). Coherent closed quasi-clique discovery from large dense graph databases. In *Kdd '06: Proceedings of the 12th acm sigkdd international conference on knowledge discovery and data mining* (pp. 797–802). ACM.

Abstract

This master thesis describes a method for the structural analysis of protein binding sites allowing the identification of functional protein families independently of sequence or fold homologies. Since binding sites are represented with graphs that illustrate their geometry and their physicochemical composition, looking for similarity between binding sites is equivalent to match their corresponding graphs. Our graph-based method used to assess the similarity between binding sites is derived from the quasi-clique detection, which is a relaxation of the clique concept, on their product graph. Its advantage is that it does not penalize the flexible nature of the protein on the 3D-level. Its use allows to enlarge the size of the maximum common substructure between proteins and thus improves their classification. Simulations on real binding sites help us to look at the impact of allowing inexact matching of the edges on the similarity assessment and the functional classification of proteins even if they do not share sequence homology.

Key words

Structural analysis, Protein binding sites, Physicochemical composition, Graph-based similarity, Quasi-clique detection, Maximum common subgraph, Functional classification.

Résumé

Ce mémoire de mastère décrit une méthode pour l'analyse structurale des sites d'interaction de protéine permettant l'identification des familles fonctionnelles indépendamment des homologies de séquence ou de repliement. Puisque ces cavités sont représentées avec des graphes qui illustrent leur géométrie ainsi que leur composition physico-chimique, la recherche de la similitude entre ces sites est équivalente au fait de mettre en correspondance leurs graphes. Notre méthode basée sur les graphes pour évaluer la similarité entre les sites d'interaction utilise la détection de quasi-cliques, qui est une relaxation du concept de clique, sur leur graphe produit. Son avantage est qu'elle ne pénalise pas la nature flexible de la protéine sur le niveau 3D. Son utilisation permet d'agrandir la taille de la sous-structure commune entre les protéines et améliore de cette manière leur classification. Les simulations sur des sites d'interaction réels nous ont permis de voir l'impact de tolerer des arcs manquants sur l'évaluation de la similarité et la classification fonctionnelle des protéines même si elles ne sont pas homologues sur le niveau séquentiel.

Mots clés

Analyse structurale, Sites d'interaction des proteines, Composition physico-chimique, Similarité basée sur les graphes, Détéction de quasi-clique, Sousgraphe maximum commun, Classification fonctionnelle.