Graph Based Pattern Discovery in Protein Structures

by
Jun Huan

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2006

Approved by,
Advisor: Wei Wang
Reader: Nikolay V. Dokholyan
Reader: Jan F. Prins
Reader: Jack Snoeyink
Reader: Alexander Tropsha

ABSTRACT
JUN HUAN: Graph Based Pattern Discovery in Protein Structures.
(Under the direction of Jan Prins and Wei Wang.)

The rapidly growing body of 3D protein structure data provides new opportunities to study the relation between protein structure and protein function. Local structure pattern of proteins has been the focus of recent efforts to link structural features found in proteins to protein function. In addition, structure patterns have demonstrated values in applications such as predicting protein-protein interaction, engineering proteins, and designing novel medicines.

My thesis introduces graph-based representations of protein structure and new subgraph mining algorithms to identify recurring structure patterns common to a set of proteins. These techniques enable families of proteins exhibiting similar function to be analyzed for structural similarity. Previous approaches to protein local structure pattern discovery operate in a pairwise fashion and have prohibitive computational cost when scaled to families of proteins. The graph mining strategy is robust in the face of errors in the structure, and errors in the set of proteins thought to share a function.

Two collaborations with domain experts at the UNC School of Pharmacy and the UNC Medical School demonstrate the utility of these techniques. The first is to predict the function of several newly characterized protein structures. The second is to identify conserved structural features in evolutionarily related proteins.

To my parents and my family.

ACKNOWLEDGMENTS

I want to thank Jan Prins and Wei Wang, who have been my thesis co-advisors over the past four years. Jan and Wei took me into the fields of data mining and bioinformatics. Wei originally recognized the importance of frequent subgraph mining techniques, which become the main theme of this dissertation. In overcoming the difficulties of graph based protein structure analysis, Jan first realized the importance of rigorous mathematical representations of protein structures. His insightful comments shaped the current thesis. It is truly because of their effort and encouragement that I am fulfilling my long term goal of using computational method to analyzing bio-molecules. In addition, Jan and Wei encouraged me during difficult times in my Ph.D. study and helped me in writing the dissertation. Finally, I thank Jan and Wei for supporting me financially through a research assistantship over my Ph.D. study, and also for a generous travel allowance that enabled me to present my work at several conferences including ACM SIGKDD'04 and '05, and IEEE CSB'06.

I also want to extend my thanks to my committee members for their comments and inputs during the thesis development. Alex Tropsha has been directly involved in the work of structure motif identification and functional annotation. Over years, Alex taught me how to capture a driving biological problem and to focus my research on delivering testable hypotheses for domain experts. I benefit from this research philosophy. Jack Snoeyink has been involved in nearly all aspects of the dissertation development, including but not limited to algorithm design, experimental design, and writing reports. Among many things that I learned from Jack, I emphasize the passion for solving intriguing mathematical problems and the skills of writing concise technical reports. Nikolay Dokholyan introduced me the complex protein structure evolutionary problem. He shared his vision and encouraged me all the way through my study of the problem with him.

I also want to give thanks to a group of faculty members that I have the privilege to work with during my Ph.D. study. My professional life benefits greatly from interactions with such a group of world experts in their fields. They are: Dr. Leonard McMillan in UNC Computer Science department, Dr. Charlie Carter in UNC Medical School, Dr. Nicolas Guex in GlaxoSmithKline, and Dr. Todd Vision in UNC Biology department. I worked with Leonard on the motifSpace project to set up a user-friendly interface. I worked with Charlie on protein structure evolution. I thank Nicolas for offering me an intern position in his group

to work on structure pattern discovery in nuclear binding proteins. I thank Todd for offering me a summer research position in his group for working on genome evolution.

I also want to thank all the students that I worked with. They are my colleagues, paper coauthors, friends, and mentors. I listed their names in a rough chronological order: Deepak Bandyopadhyay, Ruchir Shah (UNC School of Pharmacy), Angliana Washington, Jinze Liu, Xiang Zhang, Xueyi Wang, and David Williams.

I want to thank all the students in the motifSpace group in UNC Computer Science Department: Kiran Sidhu, Tao Xie, and Jingdan Zhang; I want to thank my friends at Dr. Tropsha's Molecular Modeling Laboratory at UNC School of Pharmacy: King Zhang, Jun Feng, Min Shen, and Yetian Chen; I also want to thank members of the computational geometry group: Andrea Mantler, Andrew Leaver-Fay, Yuanxin (Leo) Liu, and David O'Brien.

I thank the excellent staff of the Computer Science department at UNC Chapel Hill for their help and support. Mike Stone helped me fix my laptop several times during my Ph.D. study. Janet Jones, Tammy Pike, and Sandra Neely help me with administrative issues during my student life and being good friends.

I could not finish my Ph.D. study without financial support. I want to thank UNC General Assembly for providing my a one-year financial support. I want to thank NSF and NIH, whose grants, through PIs of Wei, Jan, Alex, and Jack, supported me.

Finally, I want to thank my wife Ke Liu and my son Stephen Huan for their supports of my work. I want to thank my parents Shuyu Huan and Chuanhui Gu for their loving and caring through my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

A protein is a chain of amino-acid molecules. In conditions found within a living organism, the chain of amino acids folds into a relatively stable three-dimensional arrangement known as the *native structure*. The native structure of a protein is a key determinant of its function [EJT04, MBHC95, OMJ$^+$97, Ric81]. Exactly how protein function is determined by protein structure is the central question in structural biology, and computational methods to compare the structures of proteins are a vital part of research in this area.

Starting from the 3D coordinates of the atoms in a protein (as obtained by a number of experimental techniques described later), *global structure comparison* can determine the similarity of two complete protein structures. Global structure comparison is widely used to classify proteins into groups according to their global similarity [HS96].

However, a protein's global structure does not always determine its function. There are well known examples of proteins with similar global structure but different functions. Conversely, there are also examples of proteins with similar function but quite different global structure. For this reason there has been increased interest in *structure pattern discovery* in order to identify structural similarity between parts of proteins [FWLN94]. The goal of structure pattern discovery is to recognize conserved geometric arrangements of amino acids in protein structures, or *structure patterns*.

This dissertation provides an introduction to the structure pattern discovery problem, focusing on novel data mining methods that detect structure patterns in protein structures. Work in this area combines computational geometry and graph theory from computer science with empirical observations and physical principles from biochemistry. The structure pattern discovery problem has important applications in identifying functional sites in proteins and function prediction of proteins, and is also of use in protein folding research and rational drug design [JT04].

1.1 Motivation

This section describes the factors that underscore the need for automated protein structure comparison methods. Readers with limited knowledge of proteins and protein structure may wish to read Chapter 2 for related background before proceeding further.

1.1.1 Rapidly Growing Catalogs of Protein Structure Data

Recognizing the importance of structural information, the Protein Structure Initiative (PSI, http://www.nigms.nih.gov/psi/) and other recent efforts have targeted the accurate determination of all protein structures specified by genes found in sequenced genomes [CBB+02, TWP+98]. The result has been a rapid increase in the number of known 3D protein structures. The Protein Data Bank (PDB) [BWF+00]), a public on-line protein structure repository, contained more than 30,000 entries at the end of year 2005. The number of structures is growing exponentially; more than 5,000 structures were added to the PDB in 2005, about the same as the total number of protein structures added in the first four decades of protein structure determination [KBD+58].

Along with individual protein structures, the structure of certain complexes of interacting proteins are known as well. While relatively few complexes have been completely determined, there is rapidly growing information about which proteins interact. Among the proteins in yeast alone, over 14,000 binary interactions have been discovered [SSK+05]. The IntAct database records 50,559 binary interactions involving 30,497 proteins [HMPL+04] from many species. Experts believe that many more interactions remain to be identified. For example, among the proteins in yeast it is estimated that there are about 30,000 binary interactions [vMHJ+03].

Additional types of data whose relation to protein structure is of interest are being accumulated as well, such as the cellular localization of proteins, the involvement of proteins in signaling, regulatory, and metabolic pathways, and post-translation structural changes in proteins [AM13, PBZ+13]. The rapidly growing body of data call for automatic computational tools rather than manual processing.

1.1.2 Pattern Discovery Aids Experiment Design

Protein structure comparison is part of a bioinformatics research paradigm that performs comparative analysis of biological data [SIK+04]. The overarching goal is to aid rational experiment design and thus to expedite biological discovery. Specifically, through comparison, the paradigm endeavors to transfer experimentally obtained biological knowledge from known proteins to unknown ones, or to discover common structure among a group of related proteins. Below we review some of the applications of local structure comparison including

pattern based structure alignment, functional site identification, structure-based functional annotation, and protein evolution. A comprehensive review can be found in [JT04].

Pattern-Based Structure Alignment

Structure alignment is vital to identifying conserved residues in protein structure, to studying the evolution of protein structures, and to facilitating structure prediction. For example, through structure alignment, domain experts discover that many diverse sequences may adopt the same global structure and such information helps significantly in structure prediction [KMS00].

Multiple structure alignment has been used to identify structural commonalities among a group of proteins. Two approaches have been investigated. The first uses multiple sequence alignment to solve the difficult alignment problem and then focuses on identifying the structural core of proteins. The second approach applies pairwise structure alignment iteratively in order to derive multiple structure alignment.

Leibowitz *et al.* first proposed a multiple structure alignment method based on recurring structure patterns [LFNW01]. In their method, multiple structure alignment is done in three steps. First, a group of common structure patterns is obtained using the geometric hashing method. Second, for each structure pattern, the occurrences of the pattern in the group of proteins are computed and aligned. Third, among many possible multiple alignments, the optimal one (one that covers the largest number of amino acid residues) is searched and used for the final multiple structure alignment.

Compared with traditional methods, pattern-based structure alignment has the following advantages [LFNW01]:

- It does not rely on multiple sequence alignment
- It handles well insertion/deletion or point mutations in protein structures
- all structures are considered simultaneously

For other pattern based structure alignment methods, see [RSKC05]

Functional Site Identification

A *functional site* is a group of amino acids in a protein that participate in the function of the protein (e.g. catalyzing chemical reactions or binding to other proteins). Identifying functional sites is critical in studying the mechanism of protein function, predicting protein-protein interaction, and recognizing evolutionary connections between proteins when there is no clear clue from sequence or global structure alignment [APG$^+$94, DW98, MPP04, VFB$^+$00]. See [TBPT05] for a recent review of known functional sites in protein structures.

Traditionally, functional sites are derived through expensive experimental techniques such as site-directed mutagenesis. This technique creates a modified protein in which one or more amino acids are replaced in specific locations to study the effect on protein function. However, site-directed mutagenesis studies are both labor intensive and time consuming, as there are many potential functional sites. In search of an alternative approach, more than a dozen methods based on the analysis of protein local structure have been developed [TBPT05]. All are based on the idea that functional sites in proteins with similar function may be composed of a group of specific amino acids in approximately the same geometric arrangement. The methods differ from each other in algorithmic details as described in Section 3. The essence of the approach is to identify local structure that recurs significantly among proteins with similar function.

Structure-based Functional Annotation

There is no question that knowing the function of a protein is of paramount importance in biological research. As expressed by George and his coauthors [GSB+05], correct function prediction can significantly simplify and decrease the time needed for experimental validation. However incorrect assignments may mislead experimental design and waste resources.

Protein function prediction has been investigated by recognizing the similarity of a protein with unknown function to one that has a known function where similarity can be determined at the sequence level [WSO01], the expression level [DLS00], and at the level of the gene's chromosome location [OMD+99].

In *structure based function annotation*, investigators focus on assigning function to protein structures by recognizing structural similarity. Compared to sequence-based function assignment, structure-based methods may have better annotation because of the additional information offered by the structure. Below, we discuss a recent study performed by Torrance and his coauthors [TBPT05] as an example of using local structure comparison for function annotation.

Torrance *et al.* first constructed a database of functional sites in enzymes [TBPT05]. Given an enzyme family, the functional sites for each protein in the family were either manually extracted from the literature or from the PSI-Blast alignment [TBPT05]. With the database of functional sites, Torrance *et al.* then used the JESS method [BT03] to search for occurrences of functional sites in the unknown structure. The most likely function was determined from the types of functional sites identified in the unknown structure. Torrance's method achieves high annotation accuracy as evaluated in several functional families.

Local Patterns Provide Evidence in Protein Evolution

There are two putative mechanisms to explain similarity between protein structures. One is *convergent evolution*, a process whereby proteins adopt similar structure and function through

different evolutionary paths [Rus98]. Convergent evolution has been studied in the serine protease family among others. Another one is *divergent evolution*, a process where proteins from the same origin become so diverse that their structure and sequence homology falls below detectable level [LPR01]. Though the exact evolutionary mechanism is still debated, studying local structure similarity can help in understanding how protein structure and function evolve.

In summary, the potential to decrease the time and cost of experimental techniques, the rapidly growing body of protein structure and structure related data, and the large number of applications necessitate the development of automated comparison tools for protein structure analysis. Next, we discuss the challenges associated with local structure comparison.

1.2 Challenges

We decompose the challenges associated with structure pattern discovery into two categories: (1) the nature of protein structure data and structure representation and (2) the computational components of structure pattern discovery.

1.2.1 The Nature of Protein Structure

In order to locate structure patterns in protein structures automatically, it is necessary to describe protein structure in a rigorous mathematical framework. To that end, we adopt the three-level view of protein structures used by Eidhammer and his coauthors in [EJT04], which is a popular view in designing pattern discovery algorithms. Another commonly used biological description of protein structure is introduced in Section 2.

Following Eidhammer's view, a protein is described as a set of elements. Common choices for the elements are either atoms or amino acids (or more precisely *amino acid residues*). Other choices are possible, see Section 4.2. Once the elements are fixed, the protein *geometry*, protein *topology*, and element *attributes* are defined. We illustrate definitions for these using amino acid residues as the protein elements.

- **Geometry** is given by the 3D coordinates of the amino acid residues, for example as represented by the coordinates of the C_α atom, or by the mean coordinates of all atoms that comprise the amino acid residue.
- **Attributes** are the physico-chemical attributes or the environmental attributes of the amino acid residues. For example, the hydrophobicity is a physico-chemical attribute of the residue. The solvent accessible surface area of an amino acid residue is an environmental attribute of the residue.
- **Topology** describes physico-chemical interactions between pairs of amino acid residues. A typical example is to identify pairs of amino acid residues that may interact through the van der Waals potential.

Structure Representations

The choice of mathematical framework for representation of a protein structure varies considerably. We review three common choices below.

- Graphs. A protein is represented as a labeled graph. A node in the graph represents an element in the protein structure, usually labeled by the attributes of the element. An edge connecting a pair of nodes represents the physico-chemical interactions between the pair of elements and may be labeled with attributes of the interaction.
- Point sets. A protein is represented as a set of points, each point represents the 3D location of an element in the protein structure. In addition, each point may be labeled with the attributes of the represented element, such as the charge, the amino acid identity, the solvent accessable area, etc.
- Point lists. A protein is represented by an ordering of elements in a point set that follows their position in the primary sequence.

All the methods are *element-based* methods since they describe a protein structure using elements in the structure. Though not commonly used, there are methods that describe a protein structure without breaking the structure into a set of elements. See [EJT04] for further details.

1.2.2 Components of Pattern Discovery

Once we have a mathematical model to describe a protein structure, the task of pattern discovery may be conveniently decomposed into a number of components. These include a definition of structure pattern and a basic notion of similarity between a structure pattern and a structure. A scoring function measures the quality of the similarity, and a search procedure uses the scoring function to search a space of potential solutions. Finally the results must be displayed in a meaningful fashion. In this section, we elaborate each of these concepts.

Defining Patterns

The way we define a structure pattern depends on the way we represent a protein structure. Three common choices for structure patterns are (1) subgraphs, (2) point subsets, and (3) point sublists. Further details of pattern definition can be found in Section 3.1.2.

Scoring Functions

Given a structure pattern and a protein structure, a *scoring function* measures the fitness of the pattern to the structure. Ideally in designing a scoring function for pattern discovery one

should take into consideration physical principles guiding protein structure. However, such physical principles take no simple form and may involve parameters that change from one protein to another. For simplicity, "generic" matching conditions such as subgraph isomorphism [APG+94] are often used. In Section 3.1.2, we survey a group of scoring functions that are used in currrent pattern discovery methods.

Search Procedures

With a scoring function, a *search procedure* is utilized to search a pattern space and identify all qualified patterns. Computational efficiency is the major concern for designing a search procedure. Our main contribution in this dissertation is a group of efficient search procedures that identify common subgraphs from an arbitrary collection of graphs. The details are discussed in Section 5.

Results Presentation

Usually the final step of pattern discovery is to present the results to end-users. One commonly used presentation method is visualization. An equally popular one is to form a hypothesis for a biological experiment. For example, recognizing the occurrence of a functional sites in a protein offers information about the possible function of the protein. Usually, both presentation methods are used after pattern discovery.

1.3 Contributions

1.3.1 Thesis Statement

Protein structure patterns that are shared by a collection of protein structures can be discovered using a labeled graph representation of protein structures and novel data mining algorithms that efficiently and effectively enumerate frequent subgraphs in the collection of protein structure graphs.

Efficiency is established by performance and scalability compared with other approaches to the problem. Effectiveness is established by the quality of the results found using these techniques. We measure the quality of the results in three applications

- Functional sites identification in proteins
- Prediction of protein enzymatic function for novel proteins
- Identification of important amino acids in protein fold families

1.3.2 Contributions

Discovering structure patterns is an important research topic in Bioinformatics with many applications. The structure pattern discovery problem may be formalized as the problem to identify common subgraphs from a group of graphs or the problem to identify common point subsets from a group of point sets. The structure pattern discovery problem under both formalizations is an NP-complete problem [SSPNW05, HWW$^+$04]. Due to this, current solutions of the structure pattern discovery problem often result in prohibitive computational cost and hence have limited utility in real world applications.

Towards a practical solution for the pattern discovery problem, this dissertation concentrates on three subproblems: (1) how to use labeled graphs to describe protein structures, (2) how to efficiently identify recurring subgraphs in a collection of labeled graphs, and (3) how applications in biology may benefit from an efficient and effective structure pattern discovery method. Below I discuss the contributions I made for each of the three subproblems.

In Chapter 4, I outline different ways of using labeled graphs to represent protein structures. In Chapter 8, a novel way to define the physical and chemical interaction of amino acid residues in a protein, based on computational geometry, is presented. In the same chapter, I present a graph representation of proteins that combines amino acid residue interactions with the geometry of the protein.

Using this graph representation of protein structures, I formalize the pattern discovery problem as a frequent subgraph mining problem and have developed several algorithms to solve the problem.

In Chapter 5, I present an efficient frequent subgraph mining algorithm FFSM (Fast Frequent Subgraph Mining) that discovers the frequent subgraphs in a collection of graphs. To solve the mining problem efficiently in FFSM, I have developed two efficient data structures. The first is termed as CAM tree (graph Canonical Adjacency Matrix tree) which is a compact representation of a space of graphs. The second is an embedding tree, which supports incremental subgraph isomorphism check for mining large graphs. As tested with synthetic and benchmark graph databases, FFSM is an order of magnitude faster than competing subgraph mining algorithms.

To minimize the number of discovered frequent subgraphs, I have developed an algorithm called SPIN (SPanning tree based mINing), presented in Chapter 6. SPIN only detects the *maximal* frequent subgraphs in a collection of graphs where a frequent subgraph is maximal if none of its supergraphs is frequent. Maximal subgraph mining significantly reduces the total number of mined subgraphs and exhibits improved efficiency.

We need to consider domain constraints in studying real-world applications. Recognizing the importance of constraints, in Chapter 7, I present a classification of constraints and show that many constraints from different real-world applications can be efficiently handled in a common framework.

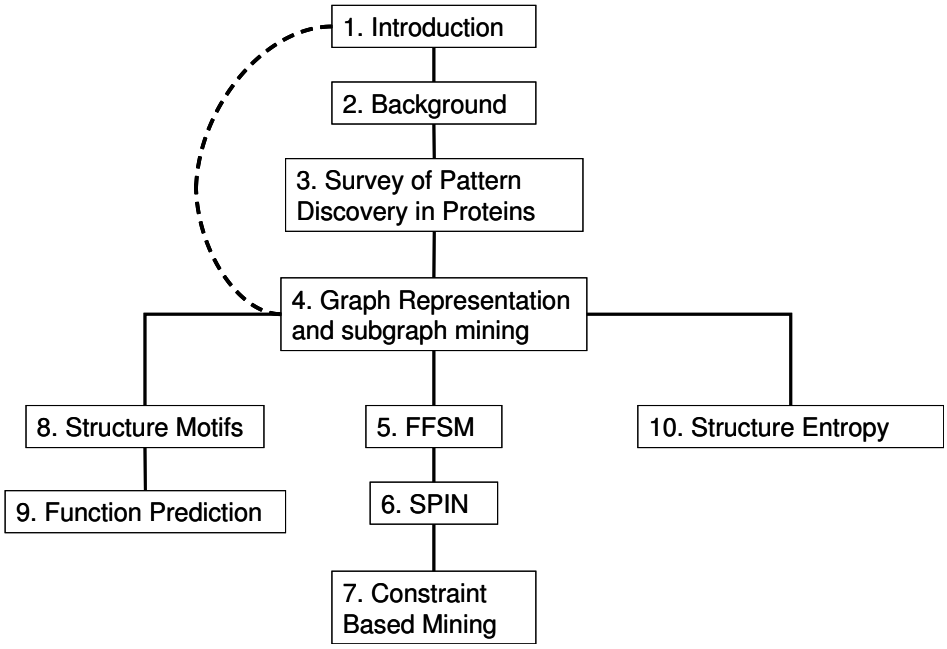

Figure 1.1: Chapter dependency in this dissertation.

To evaluate the utility of our method, I have applied my algorithms to three applications: (1) to discover functional sites in proteins, (2) to annotate protein function, and (3) to study coordinated evolution in protein sequences and structure.

In Chapter 8 I apply my graph representation of protein structure and frequent subgraph mining to obtaining structure motifs in a group of protein structures. In order to evaluate the utility of method, I closely examine four protein family whose structures have been well studied in biology. I show that patterns obtained from my method correlate well with known structure patterns.

In Chapter 9, I evaluate the prediction power of structure patterns to predict the biological function of newly characterized protein structures. I find that families with different function and similar structure can be distinguished since the structure patterns tend to identify functionally important parts of a protein automatically.

In Chapter 10, I study the problem of correlated sequence and structure evolution for proteins. I define a novel structure entropy measure to quantify structure conservation in a group of proteins. I show that the structure entropy measure correlates with both structure patterns and sequence conservation of the fold family.

Finally the dissertation contains an extensive bibliography on protein structure comparison.

1.4 Road Map of the Dissertation

Figure 1.1 explains the relationship among the remaining chapters of this dissertation. Chapters 2 and 3 are considered basic to the understanding of the remainder of the text. For readers who are familiar with the biological background and related work regarding pattern discovery in proteins, these chapters may be skipped. Chapter 4 is of primary importance in the dissertation. In Chapter 4, we set up the framework of graph representations of protein structure and apply frequent subgraph mining techniques to obtain structure patterns in proteins. Chapters 5, 6, and 7 present three methods to obtain frequent subgraphs in a collection of graphs depending on the structure patterns sought. For theoretically inclined readers, we suggest that these three chapters be studied after Chapter 4. In Chapters 8, 9, and 10 we study three applications of protein structure pattern discovery.

Chapter 2

Background

Genome sequencing projects are working to determine the complete genome sequence for several organisms. The sequencing projects have produced significant impact on bioinformatics research by stimulating the development of sequence analysis tools such as methods to identify genes in a genome sequence, methods to predict alternative splicing sites for genes, methods that compute the sequence homology among genes, and methods that study the evolutionary relation of genes, to name a few.

Proteins are the products of genes and the building blocks for biological function. Below, we review some basic background on proteins, protein structure, and protein function. See [BT91] for topics that are not covered here.

2.1 Protein Structure

2.1.1 Amino Acids

Proteins are chains of α-amino acid molecules. An *α-amino acid* (or simply an amino acid) is a molecule with three chemical groups and a hydrogen atom covalently bonded to the same carbon atom, the C_α atom. These groups are: a carboxyl group (-COOH), an amino group (-NH2), and a side chain with variable size (symbolized as R) [BT91]. The first carbon atom in a side chain (one that is connected to the C_α atom) is the C_β atom and the second one is the C_γ atom and so forth. Figure 2.1 illustrates an example of amino acids.

Different amino acids have different side chains. There are a total of 20 amino acids found in naturally occurring proteins. At physiological temperatures in a solvent environment, proteins adopt stable three-dimensional (3D) organizations of amino acid residues that are critical to their function.

2.1.2 Four Levels of Protein Structure

The levels are as follows:

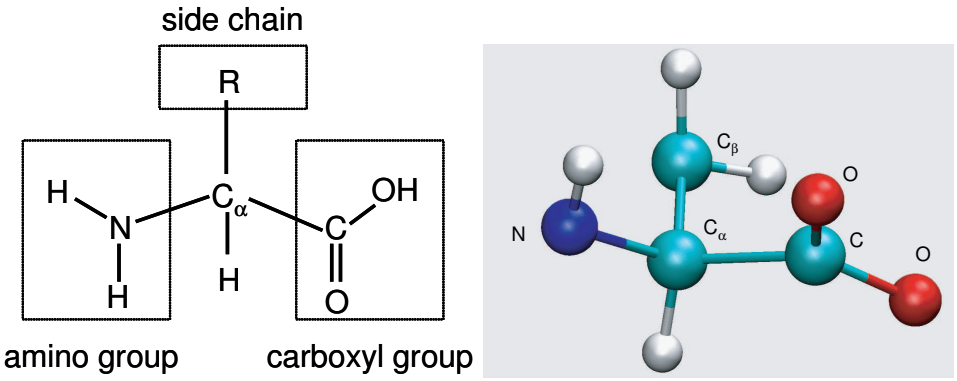

Figure 2.1: Left: a schematic illustration of an amino acid. Right: the 3D structure of an amino acid (Alanine) whose side chain contains a single carbon atom. The atom types are shown; unlabeled atoms are hydrogens. The schematic diagram is adopted from [BT91] and the 3D structure is drawn with the VMD software.

- **Primary structure** describes the amino acid sequence of a protein.
- **Secondary structure** describes the pattern of hydrogen bonding between amino acids along the primary sequence. There are three common types of secondary structures: α-helix, β-sheet, and turn.
- **Tertiary (3D) structure** describes the protein in terms of the coordinates of all of its atoms.
- **Quaternary structure** applies only to proteins that have at least two amino acid chains. Each chain in a multi-chain protein is a *subunit* of the protein and the spatial organization of the subunits of a protein is the quaternary structure of the protein. A single-subunit protein does not have a quaternary structure.

Primary Structure

In a protein, two amino acids are connected by a *peptide bond*, a covalent bond formed between the carboxyl group of one amino acid and the amino group of the other with elimination of a water molecule. After the condensation, an amino acid becomes a *amino acid residue* (or just a *residue*, for short). The C_α atom and the hydrogen atom, the carbonyl group (CO), and the NH group that are covalently linked to the C_α atom are the *main chain atoms*; the rest of the atoms in an amino acid are *side chain atoms*.

In Figure 2.2, we show the primary sequence of a protein with three amino acid residues. At one end of the sequence (the left one), the residue contains the full amino group ($-NH_3$) and is the N terminal of the sequence. The residue at the opposite end contains the full carboxyl group ($-COOH$) and is the C terminal of the sequence. By convention a protein sequence is drawn left to right from its N terminal to its C terminal.

Various protein sequencing techniques can determine the primary sequence of a protein experimentally.

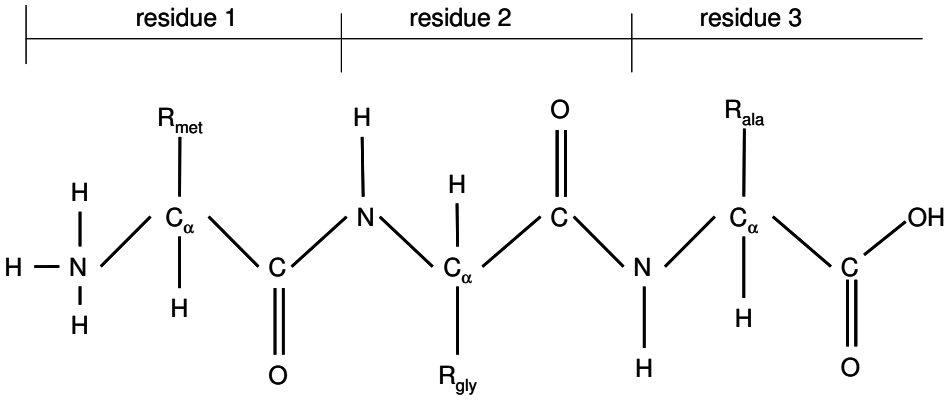

Figure 2.2: A schematic illustration of a polypeptide with three residues: Met, Gly and Ala. The peptide can also be described as the sequence of the three residues: Met-Gly-Ala.

Secondary Structure

A segment of protein sequence may fold into a stable structure called secondary structure. Three types of secondary structure are common in proteins.

- α-helix
- β-sheet
- turn

An α-helix is a stable structure where each residue forms a hydrogen bond with another one that is four residues apart in the primary sequence. We show an example of the α-helix secondary structure in Figure 2.3.

A β-sheet is another type of stable structure formed by at least two β-strands that are connected together by hydrogen bonds between the two strands. A *parallel* β-sheet is a sheet where the two β-strands have the same direction while an *anti-parallel* β-sheet is one that does not. We show examples of β-sheets in Figure 2.3.

A *turn* is a secondary structure that usually consists of 4-5 amino acids to connect α-helices or β-sheets.

Unlike the protein primary sequence, protein secondary structure is usually obtained after solving the 3D structure of the protein.

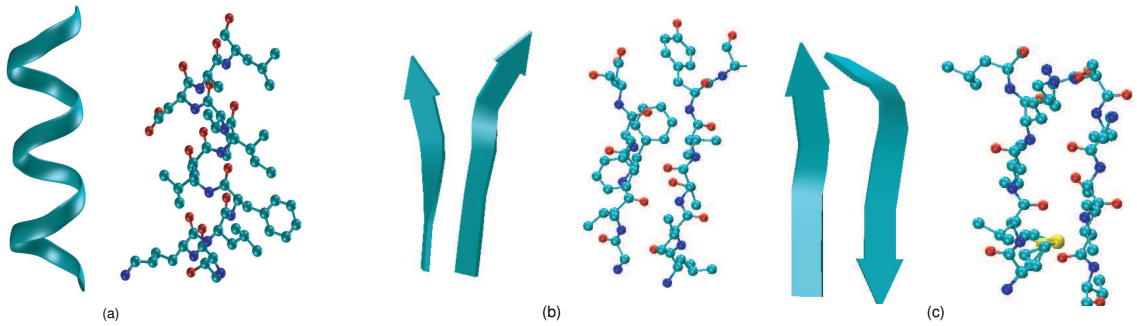

Figure 2.3: A schematic illustration of the α-helix and the β-sheet secondary structures. (a) the ribbon representation of the α-helix secondary structure (on the left) and the ball-stick representation showing all atoms and their chemical bonds in the structure (on the right). We also show the same representations for the parallel β-sheet secondary structure (b) and the anti-parallel β-sheet secondary structure (c). The α-helix is taken from protein myoglobin 1MBA at positions 131 to 141 as in [Fer99]. The parallel β-sheet secondary structure is taken from protein 2EBN at positions 126 to 130 and 167 to 172. The anti-parallel β-sheet secondary structure is taken from protein 1HJ9 at positions 86 to 90 and 104 to 108.

Tertiary Structure and Quaternary Structure

In conditions found within a living organism, a protein folds into its native structure. The tertiary structure refers to the positions of all atoms, generally in the native structure. The process of adopting a 3D structure is the *folding* of the protein. Protein 3D structure is critical for a protein to carry out its function.

In Figure 2.4, we show a schematic representation of a 3D protein structure (myoglobin). In the same figure, we also show the quaternary structure of a protein with two chains (HIV protease).

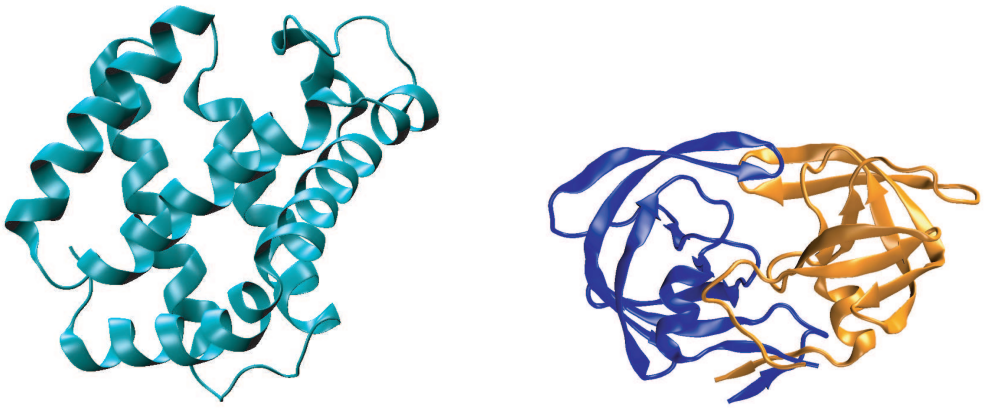

Figure 2.4: Left: The schematic representation (cartoon) of the 3D structure of protein myoglobin (PDB id: 1MBA). Right: The schematic representation (cartoon) of the 3D structure of protein HIV protease (PDB id: 1MBA). HIV protease has two chains.

Two types of experimental techniques are used to determine the 3D structure of a protein. In X-ray crystallography, a protein is first crystalized and the structure of the protein is determined by X-ray diffraction. Nuclear Magnetic Resonance spectroscopy (NMR) determines the structure of a protein by measuring the distances among protons and specially labeled carbon and nitrogen atoms [PR04]. Once the inter-atom distances are determined, a group of 3D structures (an *ensemble*) is computed in order to best fit the distance constraints.

2.1.3 Structure Classification

Domains

A unit of the tertiary structure of a protein is a *domain*, which is the whole amino acid chain or a (consecutive) segment of the chain that can fold into stable tertiary structure independent of the rest of the protein [BT91]. A domain is often a unit of function i.e. a domain usually carries out a specific function of a protein. Multi-domain proteins are believed to be the product of *gene fusion* i.e. a process where several genes, each which once coded for a separate protein, become a single gene during evolution [PR04].

Structures are Grouped Hierarchically

The *protein structure space* is the set of all possible protein structures. Protein structure space is often described by a hierarchical structure called *protein structure classification*, at the bottom of which are individual structures (domains). Structures are grouped hierarchically based on their secondary structure components and their closeness at the sequence, functional, and evolutionary level [PR04].

Here we describe a structure hierarchy, the SCOP database (Structure Classification of Proteins) [MBHC95]. SCOP is maintained manually by domain experts and considered one of the gold standards for protein structure classification. For other classification systems see [OMJ+97].

In SCOP, the unit of the classification is the domain (e.g. multi-domain proteins are broken into individual domains that are grouped separately). At the top level (most abstract level), proteins in SCOP are assigned to a "class" based on the secondary structure components. The four major classes in SCOP are:

- α domain class: ones that are composed almost entirely of α-helices
- β domain class: ones that are composed almost entirely of β-sheets
- α/β domain class: ones that are composed of alpha helices and parallel beta sheets
- $\alpha+\beta$ domain class: ones that are composed of alpha helices and antiparallel beta sheets

These four classes cover around 85% of folds in SCOP. Another three infrequently occurring classes in SCOP are: multi-domain class, membrane & cell surface domain class, and small protein domain class.

Proteins within each SCOP class are classified hierarchically at three additional levels: fold, superfamily, and family. In Figure 2.5, we show a visualization developed by the Berkeley Structural Genomics Center http://www.nigms.nih.gov/psi/image_gallery/structures.html in which globally similar structures are grouped together and globally dissimilar structures are located far away from each other. This figure shows segregation between four elongated regions corresponding to the four SCOP protein classes: α, β, α/β, and $\alpha + \beta$. Further details about protein structure classification can be found in [MBHC95].

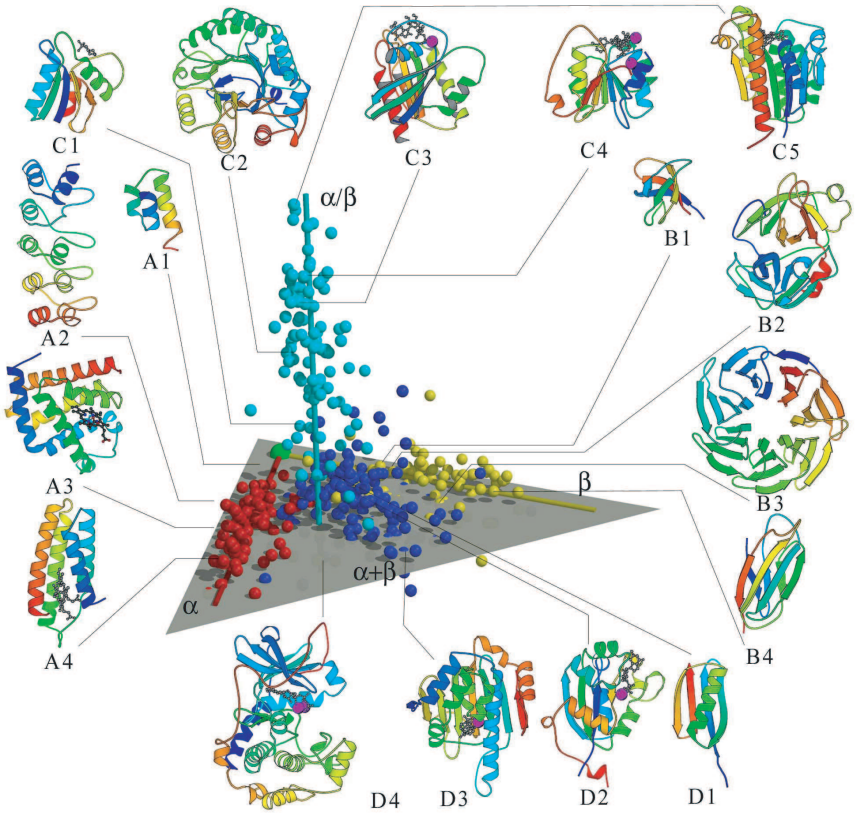

Figure 2.5: The top level structural classification of proteins based on their secondary structure components.

2.2 Protein Function

Proteins are the molecular machinery that perform the function of living organisms. Protein function can be described by the role(s) that the protein plays in an organism. Usually, protein function description is made at the molecular level, e.g. the role a protein plays in a

chemical reaction. Protein function can also be described at a physiological level concerning the whole organism, e.g. the impact of a protein on the functioning of an organism. We describe protein function at 3 different levels according to [OJT03]:

- **Molecular function**: A protein's molecular function is its catalytic activity, its binding activity, its conformational changes, or its activity as a building block in a cell. [PR04].
- **Cellular function** A protein's cellular function is the role that the protein performs as part of a biological pathway in a cell.
- **Phenotypic function**: A protein's phenotypic function determines the physiological and behavioral properties of an organism.

We need to keep in mind that protein function is context-sensitive with respect to many factors other than its sequence and structure. These factors include (but are not limited to) the cellular environment in which a protein is located, the post-translation modification(s) of the protein, and the presence or absence of certain ligand(s). Though often not mentioned explicitly, these factors are important for protein function.

In this chapter, we concentrate on the molecular function of a protein. We do so since (1) it is generally believed that native structure may most directly be related to the molecular function [GSB$^+$05], (2) determining the molecular function is the first step in the determination of the cellular and phenotypic function of a protein.

2.3 The Importance of Local Structure Comparison

Traditionally global structure comparison is well investigated in protein structure analysis. Recently the research focus has shifted towards local structure comparison. The transition from global structure comparison to local structure comparison is well supported by a wide range of experimental evidence.

2.3.1 Protein Function

It is well known that in a protein there are a few key residues that if mutated, interfere with the structural stability or the function of the protein. Those important residues usually are in spatial contact in the 3D protein structure and hence form a "cluster" in the protein structure. On the other hand, much of the remaining protein structure, especially surface area, can tolerate mutations [CS99, SKBW98]. For example, in a model protein T4 Lysozyme, it was reported that single amino acid substitutions occurring in a large fraction of a protein structure (80% of studied amino acids) tend not to interrupt the function and the folding of the protein [Mat96].

Biology has accumulated a long list of sites that have functional or structural significance. Such sites can be divided into the following three categories:

- catalytic sites of enzymes
- the binding sites of ligands
- the folding nuclei of proteins

Local structure similarity among proteins can implicate structurally conserved amino acid residues that may carry functional or structural significance [CLA99, WBL+03, DBSS00, KWK04].

Similar Structures, Different Function

It is well known that the TIM barrels are a large group of proteins with a remarkably similar fold, yet widely varying catalytic function [NOT02]. A striking result was reported in [NKGP90] showing that even combined with sequence conservation, global structure conservation may still not be sufficient to produce functional conservation. In this study, Neidhart *et al.* first demonstrated an example where two enzymes (mandelate racemase and muconate lactonizing enzyme) catalyze different reactions, yet the structure and sequence identities are sufficiently high that they are very likely to have evolved from a common ancestor. Similar cases have been reviewed in [GB01].

Different Structures, Same Function

It has been also noticed that similar function does not require similar structure. For example, the most versatile enzymes, hydro-lyases and the O-glycosyl glucosidases, are associated with 7 folds [HG99].

In a systematic study using the structure database SCOP and the functional database Enzyme Commission (EC), George *et al.* estimated 69% of protein function (at EC sub-subclass level) is indeed carried by proteins in multiple protein superfamilies [GST+04].

In summary, we want to develop methodology to recognize protein similarity at the structure pattern level, and use such method as a complemental method to ones with similar purpose that are based on sequence and global structure similarity.

Chapter 3

A Taxonomy of Local Structure Comparison Algorithms

3.1 Introduction

As discussed in Chapter 1, a structure pattern is a geometric arrangement of elements, usually at the amino acid residue level. Some other terminology also used for structure patterns includes *structure templates* [TBPT05] and *structure motifs* [EJT04]. All algorithms that identify structure patterns are generally known as the local structure comparison algorithms. Below, we introduce the common tasks in local structure comparison, the related computational components, and typical algorithms for local structure comparison.

3.1.1 Tasks in Local Structure Comparison

The task of local structure comparison is to recognize structure patterns in proteins where the patterns may be known a priori or not. When patterns are known, the recognition problem is a *pattern matching* problem in which we determine whether a pattern appears in a protein. When patterns are unknown, the recognition problem is a *pattern discovery* problem in which we find structure patterns that appear in all or many of the protein structures in a group.

The pattern discovery problem can be subdivided into two groups. The first group is *sequence-dependent* pattern discovery that discovers structure patterns in which elements have conserved sequence orders. The second is *sequence-independent* pattern discovery that discovers structure patterns without considering the primary sequence order.

For the more challenging sequence-independent pattern discovery, we subdivide the algorithms into two groups: one that detects patterns that are shared by two protein structures and one that detects patterns that occur frequently among an arbitrary group of protein structures. The taxonomy is summarized in Figure 3.1 and is discussed in detail in the subsequent sections. Before we discuss specific algorithms, we first present common components of structure pattern matching or discovery algorithms.

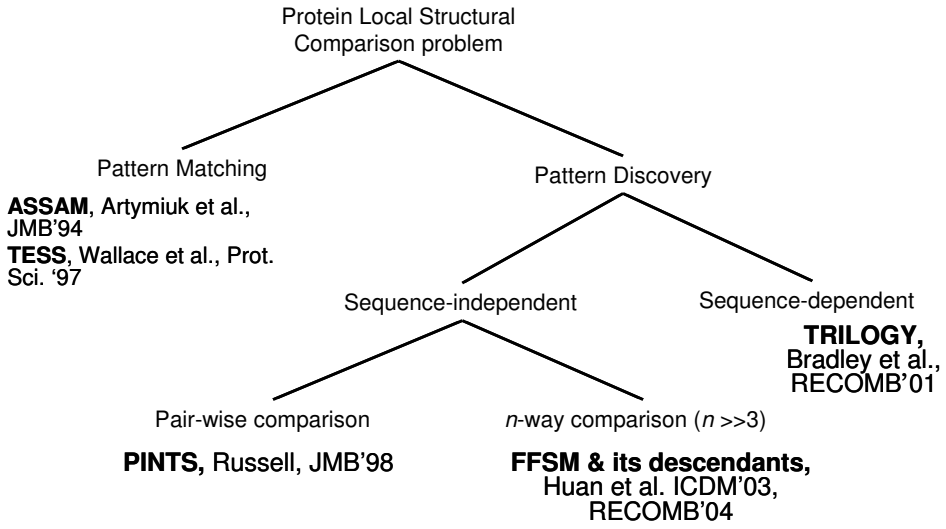

Figure 3.1: A taxonomy of local structure comparison algorithms.

3.1.2 Components

We list a group of three computational components: pattern definition, scoring function, and search procedure in order to complete the computational tasks outlined in the previous section. A pattern definition specifies the mathematical presentation of structure patterns, a scoring function measures the fitness of a pattern to a structure, a search procedure searches through a solution space and identifies the best solution(s) according to a pattern definition and a scoring function. Below we provide further detail of the three computational components.

Pattern Definition

A *structure pattern* is a geometric arrangement of structure elements, e.g. amino acids, in proteins. To provide the precise definition of structure pattern, we need to choose the elements in such definition. The typical choices are:

- atoms,
- amino acid residues,
- secondary structure elements (SSE)

The choice is usually made according to the goal of the comparison and according to the preference of the investigators. The general concern in choosing a detailed representation (e.g. atoms or amino acids) is that the coordinates of such elements in protein structures are subject to experimental noise and hence any comparison algorithms should have a high level noise tolerance. In addition, a detailed representation often leads to a more intensive

computation than a coarse representation such as SSE. On the other hand, by choosing SSEs, we may miss valuable information about a protein structure. Early structure comparison used SSE extensively, mainly for the purpose of efficient computation. Recent research tends to use amino acid or atoms because of the detailed representation.

Once we choose structure elements, there are three widely used methods to define a structure pattern.

- Graph: A node represents a structure element in the structure. An edge represents the geometry of the protein, usually in terms of pair wise element distance.
- Point set: Each point represents a structure element in the structure, with the coordinates of the element.
- Point list: the same as point set but elements are ordered as their position in the primary sequence of the protein.

Scoring Function

A *scoring function* quantifies the fitness of a structure pattern to a structure. Choosing the right scoring function plays a central role in structure comparison and involves careful design.

Two types of scoring functions are commonly used. First, subgraph matching determines whether a pattern (specified by a graph) matches a structure (specified by another graph) [Ull76].

Second, the least-square fit with rigid body transformation measures the closeness of a pattern (a point set) to a structure (another point set) [EJT04]. In least-square fit, we superimpose the pattern onto the structure such that the sum of the squared distances between corresponding elements is minimized [Kab78, Hor87]. The result of the least-square fit is conveniently expressed by a single value called the root-mean-squared-deviation (RMSD) between the pattern and the structure.

In parallel to the two methods listed previously, in matching a structure pattern with a protein structure, we may require that the structure elements in the pattern are totally ordered and the ordering is consistent with the primary sequence order of the matched elements in the protein. If we enforce this sequence order requirement, we carry out a *sequence dependent* structure comparison, otherwise, we carry out a *sequence independent* structure comparison.

As a few examples, most of the structure pattern discovery methods, such as those based on graphs [HWB$^+$04], search for common local structure patterns without enforcing the sequence order and hence are sequence independent, multi-way (or pair-wise) local structure comparison methods.

Search Procedures

Given the definition of structure patterns and a scoring function, a *search procedure* is utilized to search trough a solution space and identify the best solution (s) according to the scoring function. Computational efficiency is the major concern for designing a search procedure. There are three types of commonly used search procedures:

- Subgraph matching determines whether a subgraph occurs in a graph.
- Geometric hashing determines whether a point subset occurs in a point set.
- Frequent subgraph mining determines the group of recurring subgraphs in a collection of graphs.

The details of these search algorithms will be introduced in the following a few sections.

3.2 Pattern Matching

In pattern matching, we want to determine whether a pattern occur in a protein structure. Further more, if a pattern does occur in a protein structure, where it occurs and how many times it occurs in a protein structure. Typically, there are three types of subproblems in pattern matching [EJT04] as discussed below:

- *Occurrence* pattern matching determines whether a pattern occurs in a protein structure
- *Complete* pattern matching finds all occurrences of a pattern in a protein structure
- *Probabilistic* pattern matching calculates the probability that a pattern appears in a protein structure.

The solution of the complete pattern matching problem can be used to answer the occurrence pattern matching problem, but sometimes the latter can be computed directly in a efficient way. In the following discussion, we focus on complete pattern matching and present two algorithms: one based on subgraph isomorphism and the other one based on geometric hashing. For probabilistic pattern matching, see [AG04].

3.2.1 ASSAM

The algorithm ASSAM is one of the most successful pattern matching algorithms in local structure comparison of proteins [APG+94]. ASSAM recognizes a predefined pattern in a protein structure by transforming both the pattern and the structure to graphs and using subgraph matching to determine a possible matching(s). Below, we discuss the details of the ASSAM in terms of graph construction and subgraph matching.

Pattern Definition

ASSAM uses a graph to represent a structure pattern where

- A node in the ASSAM graph represents an amino acid residue and is labeled by the identity of the residue.
- Every pair of nodes is connected by an edge. The edge is labeled by a distance vector (to be defined) between the two residues.

In ASSAM, an amino acid residue is represented as a two-element tuple (p_1, p_2) where p_1 and p_2 are two points in a 3D space. These two points are selected to specify the spatial location and the side chain orientation of the residue and are called the "pseudo atoms" in ASSAM [1]. One of the two pseudo atoms in a residue R is designated as the "start" atom, denoted by $S(R)$, and the other is the "end" atom, denoted by $E(R)$. Refer to [APG+94] for how to obtain the pseudo atoms for the 20 different amino acids.

The *distance vector* $V_{R,R'}$ between two amino acid residues R and R' is a sequence of four distances

$$V_{R,R'} = d(S(R), S(R')), d(S(R), E(R')), d(E(R), S(R')), d(E(R), E(R'))$$

where $d(x, y)$ is the Euclidian distance of two points x and y. The distance vector is used as an edge label in the graph.

ASSAM represents structure patterns in the same way that it represents full protein structures.

Graph Matching

A distance vector V_{R_1,R_2} matches another distance vector $V_{R'_1,R'_2}$ if:

$$\begin{aligned} |d(S(R_1), S(R_2)) - d(S(R'_1)), S(R'_2))| &\leq d_{ss} \\ |d(S(R_1), E(R_2)) - d(S(R'_1)), E(R'_2))| &\leq d_{se} \\ |d(E(R_1), S(R_2)) - d(E(R'_1)), S(R'_2))| &\leq d_{es} \\ |d(E(R_1), E(R_2)) - d(E(R'_1)), E(R'_2))| &\leq d_{ee} \end{aligned}$$

where $d_{ss}, d_{se}, d_{es}, d_{ee}$ are bounds on the allowed variation in distances. These inequalities help make the matching robust against structure uncertainty in structure determination.

A structure pattern U *matches* a protein structure V, if there exists a 1-1 mapping between vertices in U and a subset of vertices in V that preserves node labels and edge labels.

[1] they are pseudo atoms since they may be located at positions that do not correspond to a real atom.

ASSAM adapts Ullman's backtracking algorithm [Ull76] to solve the pattern matching problem. We discuss the details of Ullman's algorithm for subgraph matching in Section 4.3

3.2.2 TESS

TESS models both protein structures and structure patterns as point sets where each point represent an atom in the protein structure or structure pattern. TESS determines whether a pattern matches a structure using geometric hashing [WBT97]. Specifically, the matching is done in two steps. In the *preprocessing* step, TESS builds hash tables to encode the geometry of the protein structure and the structure pattern. In the *pattern matching* step, TESS compares the contents of the hash tables and decides whether the structure pattern matches the protein structure. With minor modifications, TESS can be extended to compare a structure pattern with a group of structures. See [PA98] for other pattern matching algorithms that also use geometric hashing.

Below, we introduce the detail of the TESS algorithm.

Pattern Definition

TESS represents a structure pattern as a set of atoms $P = \{a_1, \ldots, a_n\}$ where n is the size of P. Each atom is represented by a two-element tuple $a_i = (p_i, id_i)$ where p_i is a vector in a 3D space and id_i is a nominal scalar that represents the identity of the atom.

Preprocessing in TESS

To encode the geometry of a protein structure, TESS selects three atoms from each amino acid residue and builds a 3D Cartesian coordinate system for the selection. Such 3D Cartesian coordinate system is also called a *reference frame* in TESS. For each reference frame, the associated amino acid residue is its *base* and the three selected atoms are the *reference atoms* of the frame. Predefined reference atoms exist for all 20 amino acid types [WBT97].

Given three reference atoms p_1, p_2, p_3 where each atom is treated as a point, TESS builds a reference frame $Oxyz$ in the following way:

- the origin of the $Oxyz$ system is the midpoint of the vector $\overrightarrow{p_1p_2}$,
- the vector $\overrightarrow{p_1p_2}$ defines the positive direction of the x-axis,
- point p_3 lies in the xy plane and has positive y coordinate.
- the positive direction of z-axis follows the right-hand rule.

Given a reference frame for an amino acid, TESS recomputes the coordinates of all atoms in the protein relative to this reference frame. The transformed coordinates of an atom are

discretized into an *index* that is hashed into a hash table. The associated value of an index is a two-element tuple (r, a) where r is the identifier of the base of the reference frame and a is the identifier of the corresponding atom.

TESS builds a reference frame for each amino acid residue in a protein structure and hash every atom in the protein structure into the hash table as described previously. For a protein with a total of R residues and N atoms, there are a total of $R \times N$ entries in the TESS hash table since each reference frame produces a total of N entries and there are a total of R frames.

TESS performs the same preprocessing step for a structure pattern.

Pattern Matching

For a pair of reference frames, one from a protein structure and the other one from a structure pattern, TESS determines whether there is a *hit* between the protein structure and the structure pattern. A hit occurs when each atom in the structure pattern has at least one corresponding atom in the protein structure. TESS outputs all pairs of reference frames where a hit occurs.

TESS has been successfully applied to recognize several structure patterns, including the Ser-His-Asp triad, the active center of nitrogenase, and the active center of ribonucleases, in order to predict the function of several proteins [WBT97].

3.3 Sequence-Dependent Pattern Discovery

Discovering common structure patterns from a group of proteins is more challenging than matching a known pattern with a structure. Here we introduce two algorithms: TRILOGY [BKB02] and SPratt [JET99, JECT02] that take advantage of sequence order (and separation) information of amino acid residues in a protein structure to speed up pattern discovery. Patterns identified by these methods are *sequence-dependent* structure patterns.[2]

3.3.1 TRILOGY

TRILOGY identifies sequence-dependent structure patterns in a group of protein structures [BKB02]. There are two phases in TRILOGY: initial pattern discovery and pattern growth. Before we discuss the two phases in detail, we present the pattern definition and matching condition used in TRILOGY.

[2]Amino acid residues in sequence-dependent patterns are in sequence order but not necessarily consecutive in the sequence.

Pattern Definition

In TRILOGY, a three-residue pattern (a triplet) P is a sequence of amino acid residues and their primary sequence separations such that

$$P = R_1 d_1 R_2 d_2 R_3$$

where R_i ($i \in [1,3]$) is a list of three amino acid residues sorted according to primary sequence order in a protein and d_i ($i \in [1,2]$) is the number of residues located between R_i and R_{i+1} along the primary sequence (the *sequence separation*).

Each residue R in TRILOGY is abstracted by a three-element tuple (p, v, id) where p is a vector specifying the 3D coordinates of the C_α atom in R, v is the vector of $C_\alpha C_\beta$ atoms, and id is the identity of the residue.

Pattern Matching

A triplet $P = R_1 d_1 R_2 d_2 R_3$ matches a protein structure if there exists a triplet $P' = R'_1 d'_1 R'_2 d'_2 R'_3$ in the structure such that

- (1) the corresponding amino acid residues (R_i and R'_i, $i \in [1,3]$) have similar amino acid types,
- (2) the maximal difference between the corresponding sequence separations $|d_i - d'_i|$ $i \in [1,2]$ is no more than a specified upper-bound (e.g. 5),
- (3) the geometry of two triplets matches. This suggests that:
  - the difference between the related C_α-C_α distances is within 1.5 Å
  - the angle difference between two pairs of matching C_α-C_β vectors is always within 60°

If a protein satisfies condition (1) and (2) but not necessarily (3) it is a *sequence match* of the triplet P. If a protein satisfies condition (3) but not necessarily (1) or (2) it is a *geometric match* of the triplet P. By definition, a protein matches a triplet P if there is a sequence match *and* a geometric match to P.

The pattern definition and matching condition for larger patterns with d amino acids are defined similarly to the above, but use $2d-1$ element tuples instead of triples.

Triplet Discovery

TRILOGY takes as inputs a group of protein structures and produces a sequence alignment of the structures using information provided by the HSSP database [SS91].

After sequence alignment, all possible triplets are discovered. For each triplet, TRILOGY collects two pieces of information: the total number of sequence matches and the total number of structure matches, and assigns a score to the triplet according to a hypergeometric distribution. Only highly scored triplets are used to generate longer patterns.

Pattern Growth

If a highly scored triplet shares two residues with another triplet, the two patterns are "glued" together to generate a larger pattern with four amino acid residues in the format of $R_i d_i R_4$ where $\{R_i\}, i \in [1,4]$ and $d_i, i \in [1,3]$ are defined similarly to ones in triplets. Longer patterns in TRILOGY are generated similarly.

3.3.2 SPratt

Like TRILOGY, the SPratt algorithm also uses the primary sequence order information to detect common structure patterns in a group of protein structures [JET99, JECT02]. Unlike TRILOGY, SPratt discards the requirement that the sequence separation between two residues should be conserved. In the following discussion, we present the details of the SPratt algorithm.

Pattern Definition

In SPratt, a pattern P is a list of amino acid residues

$$P = p_1, \ldots, p_n$$

where n is the length of P. Each residue in SPratt is abstracted as a two-element tuple (p, id) where p is a vector representing the coordinates of the C_α atom in R and id is the identity of the residue. Additional information such as the secondary structure information and the solvent accessible area may be included to describe a residue.

Pattern Matching

A pattern P of length n matches with a protein structure Q if we can find a sequence of amino acid residues $S = s_1, \ldots, s_n$ sorted according to the primary sequence order in Q such that

- the residue identity of s_i matches with the residue identify of p_i, $i \in [1,n]$.
- the root-mean-squared-deviation (RMSD) value of the corresponding locations in P and S is below some threshold.

Pattern Discovery

Pattern discovery in SPratt is done in three steps. First, SPratt picks an amino acid residue and selects all neighboring residues within a cutoff distance. It converts the set of neighboring amino acid residues into two strings, called *neighbor strings*: one that includes all residues that precede the target residue in the sequence and the second that includes all residues that follow. Both strings are sorted according to the primary sequence order. For each amino acid residue and each protein structure in a data set, SPratt computes the neighbor strings and puts all the strings together. Encoding neighboring residues in this way, the neighbor strings reflect the primary sequence order but not the separation between any residues.

Second, the Pratt string matching algorithm [Jon97] is used to identify all sequence motifs that occur in a significant part of the data set.

Third, for each sequence motif, the geometric conservation of the motifs (measured by the pairwise RMSD distance between all the instances of the sequence motif) is evaluated. SPratt selects only those with significant geometric conservation.

3.4 Pairwise Sequence-Independent Pattern Discovery

In the previous section, we discussed algorithms that identify sequence-dependent structure patterns. In this section and the one that follows, we discuss algorithms that identify structure patterns without the constraint of sequence order, or *sequence-independent* structure patterns.

We divide sequence-independent structure pattern discovery algorithms into two groups according to whether they work on a pair of structures or on an arbitrary collection of structures. In this section, we review pairwise sequence-independent pattern discovery methods and in the next section we show how pairwise comparison can be extended to multiway comparison of protein structures. Pairwise sequence-independent pattern discovery methods include:

- Geometric hashing methods that represent protein structures as point sets and use geometric matching to find structure patterns [NW91, FWLN94].
- Graph matching methods that model protein structures as labeled graphs and perform subgraph matching to detect conserved patterns [GARW93, MSO03, SSR04, SAW03, WTR$^+$03].

3.4.1 Geometric Hashing

This class of methods models a protein structure as point sets and use the geometric hashing technique to obtain common point subsets from two structures. There is no fundamental difference in applying geometric hashing for pairwise structure pattern identification and that

of pattern matching as exemplified by the TESS algorithm in Section 3.2.2. Rather than repeating the discussion of preprocessing and geometric matching that are common to almost all geometric hashing based methods, we present an analysis of computational complexity. We also show how different techniques may reduce the asymptotic complexity of the computation.

Pattern Definition

A structure is represented as a set of amino acid residues $P = \{a_1, \ldots, a_n\}$ where n is the size of P. Each residue is represented by a two-element tuple $a_i = (p_i, id_i)$ where p_i is a vector in a 3D space that represents the spatial location of the residue (e.g. the coordinates of the C_α atom) and id_i is the identity of the residue.

This definition was originally used by Nussinov & Wolfson [NW91]. The complexity of preprocessing a single protein structure with n residues is bounded by $O(n^4)$. This is because there are a total of $\binom{n}{3}$ triplets in a protein. For each triplet we build one reference frame. For each reference frame, we compute the new coordinates of all n residues in the protein according to the frame. The complexity of this preprocessing step is hence $n \cdot O(\binom{n}{3}) = O(n^4)$.

At the matching stage, two structures are preprocessed and the results are stored in a single hash table. After preprocessing, we scan the hash table once to report the shared structure patterns. Clearly, the post processing step is bounded by the total number of entries in the hash table which is itself bounded by $O(n^4)$. Therefore the overall computational complexity is $O(n^4)$.

Nussinov & Wolfson present an algorithm to speed up the computation from $O(n^4)$ to $O(n^3)$. In the improved version, rather than using a triplet to build a reference framework, two points are used to build a reference framework. There are a total of $O(n^2)$ point pairs in a data set with n points and hence the overall complexity is reduced to $O(n^3)$.

Fischer *et al.*[FWLN94] proposed a more efficient algorithm with complexity $O(n^2)$. For a protein structure with n residues, rather than building a total of $O(n^3)$ reference frames, Fischer's method builds a total of n reference frames. This is done by always picking up three residues that are consecutive in the primary sequence and building one reference frame for each such triplet. There are a total of $O(n)$ such triplets so the overall complexity is $O(n^2)$.

Geometric hashing has been applied to recognize local structure similarity for proteins even if they have globally different structures [FWLN94].

3.4.2 PINTS

There is a group of methods that utilize graph theory to model protein structure and use subgraph isomorphism to detect recurring patterns among a pair of protein structures [SR03, MSO03, SKK02]. In such methods, a protein structure is modeled by a graph where each node models an amino acid residue, labeled by the residue identity, and each edge connects

a pair of residues, labeled by a variety of information related to the geometry of the protein and the physico-chemical interactions between the pair of residues.

Below we review PINTS algorithm [Rus98, SSR98], which takes as input two protein structures and identifies all structure patterns common to the two structures in detail. For related methods, see [GAW97, MSO03, SKK02, WKHK04].

Pattern Definition

PINTS uses a graph to represent a structure pattern where

- A node in the PINTS graph represents an amino acid residue and is labeled by the identity of the residue.
- Two nodes are connected by an edge labeled by the distance vector (to be defined) between the two residues.

In PINTS, an amino acid residue R is a three-element tuple (p_1, p_2, p_3) that represents the C_α atom, the C_β atom, and a functional atom in the residue R. One functional atom is defined for each of the 20 amino acid residue types.

A *distance vector* between two residues R_1, R_2 in PINTS is a three-element tuple V

$$V = (d_\alpha^{R_1,R_2}, d_\beta^{R_1,R_2}, d_f^{R_1,R_2}) \tag{3.1}$$

where $d_\alpha^{R_1,R_2}, d_\beta^{R_1,R_2}, d_f^{R_1,R_2}$ are the (Euclidian) distances between the C_α atom, the C_β atom, and the functional atom in the side chain of the two residues.

Graph Matching

The distance vector V_{R_1,R_2} matches the distance vector $V_{R_1',R_2'}$ if

$$\begin{aligned} &\mid d_\alpha^{R_1,R_2} - d_\alpha^{R_1',R_2'} \mid \leq d_\alpha \\ &\mid d_\beta^{R_1,R_2} - d_\beta^{R_1',R_2'} \mid \leq d_\beta \\ &\mid d_f^{R_1,R_2} - d_f^{R_1',R_2'} \mid \leq d_f \end{aligned}$$

where d_α, d_β, d_f are predefined tolerances. PINTS uses values 7.5 Å, 6.6 Å, and 6 Å, respectively.

A structure pattern P *matches* a structure Q if there exists 1-1 mapping of residues in P to a set of residues in Q such that corresponding nodes have identical node labels and corresponding edges are labeled by matching distance vectors.

Pattern Discovery

PINTS uses a modified Ullman's subgraph isomorphism test to identify all shared subgraphs of two graphs. An overview of the Ullman's subgraph isomorphism algorithm can be found in Section 4.3.

The statistical significance of identified patterns is estimated using a sophisticated model [SSR98], which involves the RMSD between the two instances of the patterns, the number of residues in the pattern, the abundance of those residues, and their connectivity along the sequence.

Many interesting patterns have been identified by the PINTS method including the serine protease active center, the NAD binding motif in NAD binding proteins, and binding pockets of chorismate mutases.

3.5 Multiway Sequence-Independent Pattern Discovery

In this section, we present a review of sequence-independent patten discovery methods that work on a group of two or more structures. These methods are:

- Delaunay Tessellation
- Geometric hashing
- Frequent subgraph mining

3.5.1 Delaunay Tessellation

This class of methods [KT03, CCT02, TCCV03] identifies local structure patterns based on the Delaunay Tessellation technique.

Delaunay tessellation partitions a structure into an aggregate of non-overlapping, irregular tetrahedra that identify the nearest-neighbor residue quadruplets for any protein. The decomposition is unique and can be made robust in the presence of uncertainty of the residue positions [BS04]. Recurring structural patterns can be identified from tetrahedra that occur in many structures. Studies have explored the hypothesis that four-residue packing motifs can be defined as structure specific residue signatures and can be utilized in the annotation of structural and functional classes of protein structures [TCCV03]. There are also studies that identify structure patterns based on the analysis of protein structures in a family represented as a network of residue contacts obtained by Delaunay tessellation [CCT02, HWW$^+$04].

3.5.2 Geometric Hashing

Recently geometric hashing has been applied to perform multiple structure alignment [LFNW01] and to identify functional sites in protein structures [SPNW04, SSPNW05]. It has

been also applied to atom-level representations of protein structures [SSPNW05].

The extension of geometric hashing methods to find common structure patterns among many structures [SPNW04, SSPNW05] suffer from limited scalability since such methods have exponential running time in the total number of structures.

3.5.3 Clique Detection

Wangikar *et al.* compared multiple structures by first performing a pairwise comparison and then postprocessing the pairwise results for multiway comparison [WTR$^+$03].

In their research, Wangikar *et al.* compute structure patterns in three steps: (1) using a pair-wise comparison method [Rus98], they identify all common structure patterns from each pair of proteins in a data set. (2) A graph $G = (V, E)$ is built for the derived patterns where a node in G is a pattern and an edge connect two nodes if (a) the "distance" of the two patterns is within a certain threshold and (b) the two patterns are from different proteins. In their study, the RMSD metric is used to measure the distance between two patterns. In the graph, a clique with size k corresponds to a recurring pattern that occurs in k proteins in the original set. In step (3) Wangikar *et al.* applied an algorithm to search for all maximal cliques with large size.

Wangikar *et al.* have successfully applied their method to a total of 17 protein families from the SCOP database to locate functional sites in those families including serine protease, EF-hands proteins, cupredoxins, and glutathione-S-transferases.

3.5.4 Frequent Subgraph Mining

In frequent subgraph mining, a protein structure is represented by a graph. Given a group of graphs and a matching condition (usually specified as subgraph isomorphism), the goal of frequent subgraph mining is to discover all frequent subgraphs in the collections of graphs [YH02, HWP03]. We discuss frequent subgraph mining algorithms in detail in the next two chapters. These methods have excellent scaling behavior as the number of structures increases.

Chapter 4

Pattern Discovery Using Graph Mining

Graph is a ubiquitous data representation that has been used in many application domains for different real-world objects. Examples of graph representations include the topology of communication networks, social networks, citation networks, chemical 2D structures, protein 3D structures, RNA structures, gene phylogeny data, protein-protein interaction data, and signaling, regulatory, and metabolic pathways. An an example that have been extensively studied in the pharmaceutical industry, the 2D structure of a chemical can be modeled as an undirected labeled graph where each node represents an atom in the chemical, labeled by the atom type, and each edge corresponds to a chemical bond, labeled by the bond type. With graph representations, automated classifiers have been built to identify target chemicals in a database of known chemicals [BB02].

Graphs have also been widely utilized for representing protein structure in protein structure comparison [APG+94]. In the following discussion, we first give a formal definition of labeled graphs (graphs with node and edge labels) and then discuss two methods that use graphs to represent protein structures. A more sophisticated method developed in this dissertation, which combines existing graph representations of protein structures, is discussed in Chapter 8.

4.1 Labeled Graphs

4.1.1 Labeled Simple Graphs

Here we first define labeled simple graphs and then labeled multigraphs and peudographs.

Definition 4.1.1 *A **labeled simple graph** (graph) is a quadruple $G = (V, E, \Sigma, \lambda)$ where V is a set of vertices or nodes and $E \subseteq V \times V$ is a set of edges joining two distinct nodes. Σ is the set of nodes and edge labels and $\lambda : V \cup E \rightarrow \Sigma$ is a function that assigns labels to nodes and edges.*

The *size* of a graph G, denoted by $|G|$ is the cardinality of its node set. The *degree* of a node v is the number of edges incident with v. We use $V[G]$ and $E[G]$ to denote the set of nodes and edges for a graph G, respectively. We usually assume node labels and edge labels are disjoint and a total ordering is defined for the label set Σ. A *graph database* is a list of labeled graphs where each graph is assigned an integer identifier called *graph id*. A simple graph G is *undirected*, if the binary relation $E[G] \subset V \times V$ is symmetric(i.e. $(u, v) \in E[G]$ implies $(v, u) \in E[G]$ and $\lambda(u, v) = \lambda(v, u)$), otherwise, G is *directed*. Unless stated otherwise, all graphs are undirected in this dissertation.

4.1.2 Multigraphs and Pseudographs

A *multigraph* is a graph where a pair of nodes may be joined by more than one edge (multiple edges). A *loop* is a degenerate edge that joins a node to itself. A simple graph can have neither loops nor multiple edges, a multigraph may have multiple edges but no loops, and a *pseudograph* may have both. To make the description precise, we have the following definition.

Definition 4.1.2 *A **labeled multigraph** is a quadruple $G = (V, E, \Sigma, \lambda)$ where $\lambda : V \cup E \rightarrow 2^{\Sigma} - \{\emptyset\}$ is a function that assigns (multiple) labels to nodes and edges. 2^{Σ} is the powerset of a set Σ. The interpretations of V, E, and Σ are the same as those of simple graphs. If a labeled multigraph contains a loop, it is a labeled pseudograph.*

Example 4.1.1 *In Figure 4.1, we show a graph database with three graphs P, Q, and S with graph id 10, 20, and 30, respectively. The edge (p_2, p_5) in graph P has multiple labels $\{x, y\}$ and hence P is not a simple graph. Graphs Q and S are simple graphs.*

We notice that simple graphs are special cases of multigraphs. Below, we use the term "multigraph" in such way that it may refer to both simple graphs or graphs that indeed contain multiple labels for a node or an edge. Throughout our discussion, we use capital letters to represent graphs and lower case letters with subscripts to denote nodes in graphs. The order of nodes in a graph is arbitrary.

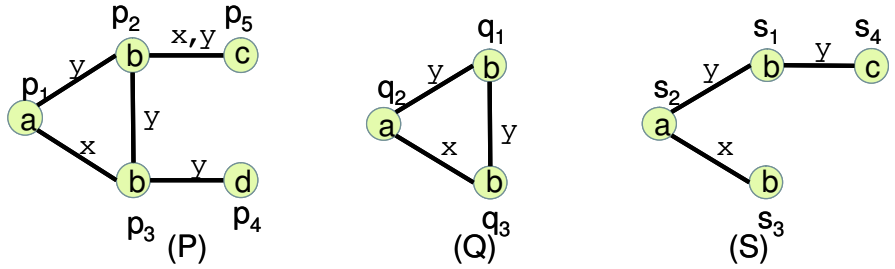

Figure 4.1: A database $\mathcal{G}$ of three labeled graphs. The labels of nodes and edges are specified within the nodes and along the edges.

4.1.3 Paths, Cycles, and Trees.

We also use the following graph-related terms:

- A *simple path* (path) is an n-node undirected graph L where $V[L] = \{l_i\}$, $i \in [1, n]$ and $E[L] = \{(l_i, l_{i+1})\}$, $i \in [1, (n-1)]$. $n > 0$ is the *length* of the path L.
- A graph G is *connected* if for each pair of distinct nodes (u, v), there exists a path $L \subseteq G$ such that $l_1 = u$ and $l_n = v$ where n is the length of L.
- A *cycle* O is an n-node path L with one additional edge connecting l_1 and l_n. n is the *length* of O.
- An *acyclic* graph is a graph with no cycle.
- A *tree* is a connected acyclic graph.

4.2 Representing Protein Structures with Graphs

Graphs have been widely used to represent protein structures. In such representations, we generally use a node to represent an amino acid residue and an edge to represent the binary relation between a pair of residues. Depending on the applications, the binary relation may be (Euclidian) distances between pairs of amino acid residues (distance matrix) or the physico-chemical contacts between residues (contact maps). Below we discuss the detail of distance matrix and contact map of protein structures.

4.2.1 Distance Matrix

A matrix $X_{n,n} = (x_{i,j})$ $(1 \leq i, j \leq n)$ is the distance matrix for a protein P with n elements, if the entry $x_{i,j}$ is the (Euclidian) distance of the ith and jth element in protein P. For each protein structure, there is exactly one distance matrix. However, the reserve is not true. Given a distance matrix X, there are at most two structures corresponding to the matrix (when $n > 4$). This is because inter-element distances are the same for a mirror image of a structure.

Using a distance matrix at the amino acid residue level, a protein structure is represented by a graph where each node represents a residue and each edge connects a pair of residues. Edges in such graphs are labeled by the discretized distances between the two residues that they connect.

4.2.2 Contact Map

A protein contact map is the same as the protein distance matrix representation, except that edges in contact maps are not labeled by distances but by a boolean indicating whether

the pair of amino acid residues are in "contact" or not. There are many ways to define the "contact" relation. The most common way is a distance based method where a pair of residues are in contact if their distance is below a distance threshold and otherwise not [HSS+02]. More sophisticated methods such as Delaunay Tessellation and almost-Delaunay are also used to define the residue contact relation [HWW+04].

4.3 Subgraph and Subgraph Isomorphism

A fundamental part of pattern discovery in graph represented real-world objects is to define what is a pattern and to specify how to decide whether a pattern occurs in a graph represented object. To make this more precise, we use the follow definitions.

Definition 4.3.1 *A multigraph G is a **subgraph** to another multigraph G' if:*

- $\forall u \in V[G], (u \in V[G'])$,
- $\forall u, v \in V[G], ((u, v) \in E[G] \Rightarrow (u, v) \in E[G'])$,
- $\forall u \in V[G], (\lambda(u) \subseteq \lambda'(u))$, *and*
- $\forall (u, v) \in E[G], (\lambda(u, v) \subseteq \lambda'(u, v)$.

It is easy to see that the above definition applies to simple graphs also.

In this dissertation, a *pattern* is a graph. We say a *pattern* occurs in a graph G, if it is topologically identical to a subgraph of G. To make this more precise, we introduce the following definition:

Definition 4.3.2 *A multigraph G is **subgraph isomorphic** to another multigraph G' if there exists a 1-1 mapping $f : V[G] \rightarrow V[G']$ such that:*

- $\forall u \in V[G], (\lambda(u) \subseteq \lambda'(f(u)))$,
- $\forall u, v \in V, ((u, v) \in E[G] \Rightarrow (f(u), f(v)) \in E[G'])$, *and*
- $\forall (u, v) \in E[G], (\lambda(u, v) \subseteq \lambda'(f(u), f(v)))$.

G' in the above definition is a *supergraph* of G. The bijection f is a *subgraph isomorphism* from G to G' and the node image $f(V[G])$ of V is an *occurrence* of G in G'. With a slight abuse of notation, we may use the term "subgraph" to refer to a "subgraph isomorphic" relation. Two graphs G and G' are *isomorphic*, denoted by $G = G'$ if they are mutually subgraphs of each other. Non-isomorphic subgraph G of G' is a *proper subgraph* of G', denoted by $G \subset G'$. A *proper supergraph* is defined similarly.

An induced subgraph is one that preserves all edges in the larger graph. In other words, a graph G is *induced subgraph isomorphic* to another graph G' if $G \subseteq G'$ with a bijection $f : V[G] \rightarrow V[G']$ such that (1) $\forall(u, v) \in E[G] \Leftrightarrow (f(u), f(v)) \in E[G']$ and (2) $\lambda(u, v) = \lambda'(f(u), f(v))$. We call a graph G an *induced subgraph* of G' if G is induced subgraph isomorphic to G'.

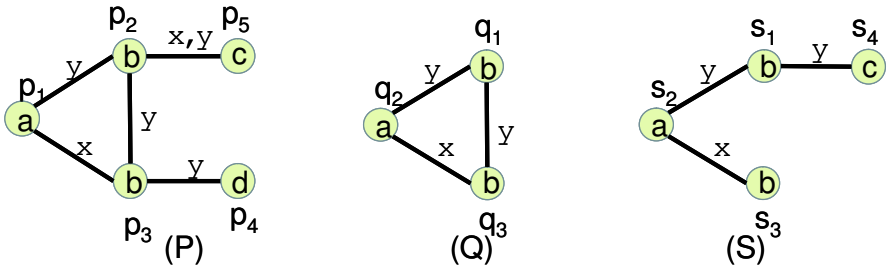

Figure 4.2: A database $\mathcal{G}$ of three labeled graphs duplicated from Figure 4.1. The label(s) of nodes/edges are specified within the nodes/along the edges.

Example 4.3.1 *In Figure 4.2, we show three graphs that are duplicated from Figure 4.1 for the readers' convenience. The function $f : q_1 \rightarrow p_2$, $q_2 \rightarrow p_1$, and $q_3 \rightarrow p_3$ is a subgraph isomorphism from graph Q to P and hence Q occurs in P. The set $\{p_1, p_2, p_3\}$ is an occurrence (and the only one) of graph Q in P. We notice that Q is also an induced subgraph of P since Q preserves all edges of P in the node image $\{p_1, p_2, p_3\}$. Similarly, S occurs in P but S is not an induced subgraph of P.*

4.3.1 Ullman's Algorithm

Ullman's algorithm is one of the most widely used algorithms to solve the subgraph isomorphism problem [Ull76]. Though Ullman originally developed the algorithm for unlabeled and undirected graphs, this algorithm can be easily extended to many other types of graphs with little extra effort regardless of whether these graphs are labeled or unlabeled, have multiple edges or not, have graph loops or not, and are directed or undirected. In the following discussion, we present the basic form of Ullman's subgraph isomorphism algorithm for unlabeled and undirected graphs. See [HBW+05] if interested in subgraph isomorphism in other types of graphs.

In Ullman's algorithm, the pattern graph and graph to be matched with (the parent graph) are represented by standard adjacency matrices $A_{n,n}$ and $B_{m,m}$ where n and m are the total numbers of nodes in graphs A and B respectively. $a_{i,j}$ equals to 1 if the ith node of the graph A is connected to the jth node of A and otherwise 0. Matrix $B_{m,m}$ is defined the same way as that of $A_{n,n}$.

Ullman used an $n \times m$ binary matrix M, known as the *permutation matrix*, to record intermediate search results. Each row in the permutation matrix has exactly one 1 and each

column has at most a single 1. The permutation matrix encodes a 1-1 mapping from nodes of A to those of B. To see this clearly, we interpret an entry $m_{ij} = 1$ in M as a match between the ith node in A and the jth node in B. Since each row of M has exactly one 1, each node in A maps to exactly one node in B; since each column of M has at most a single 1, no two nodes in A can match the same node in B. In other words, M encodes a 1-1 mapping from nodes of A to those of B.

Using linear algebra, we obtain $C = M(MB)^T$ where X^T is the transpose of matrix T. One important theorem about graph matching is that M stands for a subgraph isomorphism from A to B, if and only if:

$$\forall(i, j : 1 \leq i, j \leq n), (a_{ij} = 1 \Rightarrow c_{ij} = 1) \tag{4.1}$$

To search for all successful matches, Ullman's algorithm enumerates the space of all possible permutation matrices M using a backtrack method. The proof the theorem and the algorithmic details of the backtrack search can be found in [Ull76].

4.4 A Road Map of Frequent Subgraph Mining

Because graphs are ubiquitous data types in many applications including protein structure analysis [HWP03, HWB$^+$04], identifying recurring patterns of graphs has attracted much recent research interest. Recurring subgraph patterns provide insights of the underlying relationships of the objects and are the starting point for subsequent analysis such as clustering and classification. Successful applications of recurring subgraph pattern identification include improving storage efficiency of databases [DFS99], efficient indexing [GW97, SBV01], and web information management [Zak02a, RGM03]. With no surprise, algorithms for graph based modeling and analysis are going through a rapid development [HWB$^+$04].

In this section we define the frequent subgraph mining problem, followed by an introduction to current algorithms of mining graph databases.

4.4.1 The Problem

Given a set Σ, the *graph space* G^* is all possible simple connected graphs with labels from Σ. Given a group of graphs $\mathcal{G} \subseteq G^*$, the *support* of a simple graph G, denoted by $s(\mathrm{G})$, is the fraction of $\mathcal{G}$ where G occurs.

The frequent subgraph mining problem is defined as:

Definition 4.4.1 *Given a graph database $\mathcal{G}$ and a parameter $0 < \sigma \leq 1$, the **frequent subgraph mining** problem is to identify all simple graphs $G \in G^*$ such that the support of G is at least σ.*

An algorithm that solves the frequent subgraph mining problem is a *frequent subgraph mining algorithm*. We focus on simple graphs in frequent subgraph mining for historical reasons: most of the current algorithms only identifies simple graphs. In addition, most of the algorithms assume that the input graph database contains only simple graphs. One notable exception for handling multigraphs in presented in [HPW+06]. We consider only connected graphs in a graph space since unconnected graphs can be viewed as a group of connected graphs. Once connected frequent subgraphs are identified, unconnected ones can be obtained using frequent item set mining techniques, as observed in [KK01].

4.4.2 Overview of Existing Algorithms

Since frequent subgraph mining is computationally challenging, early research focused on either approximation techniques such as SUBDUE [HCD94] or methods that are only applicable for small databases like Inductive Logic Programming [DTK98].

Because computing frequent subgraphs in large graph databases is time consuming, research in frequent subgraph mining focuses on improving the efficiency of the algorithms. All scalable algorithms take advantage of the *anti-monotonicity* of frequency, which asserts that any supergraph of an infrequent subgraph pattern remains infrequent. We can break existing frequent subgraph mining algorithms into three components that are discussed in the sequel.

- Searching for initial seeds: preprocessing the input graph database and identifying a set of initial frequent subgraph patterns as "seeds". Graph topology of seeds is usually simple, e.g. single edge or simple paths.
- Proposing candidate subgraphs: for each seed, a new set of patterns are proposed that are supergraphs of the seed and are likely to be frequent.
- Validating candidate subgraphs: for each proposed candidate, the support value is computed. Only frequent ones are selected as seeds for the next iteration.

Components (2) and (3) may be utilized repeatedly in order to obtain all frequent subgraphs.

Below, we divide existing frequent subgraph mining methods into three groups based on how candidates are proposed.

- Edge based methods: generate new subgraphs by adding one edge to existing frequent subgraphs
- Path based methods: decompose a graph into a set of paths and increase graph patterns by adding a path at a time
- Tree based methods: first identify all frequent tree patterns and then discover cyclic graph patterns.

There are other types of graph mining algorithms that focus on mining a useful subset of frequent subgraphs. For example, maximal frequent subgraph mining [HWPY04] identifies only those frequent subgraphs for which none of their supergraphs is frequent. Coherent subgraph mining uses mutual information to select subgraphs that may be infrequent in an overall data set [HWW+04]. For a more recent review of different subgraph mining algorithms, see [HWPY04].

4.4.3 Edge Based Enumeration

Level-wise Search: the FSG Algorithm

FSG (Frequent Subgraph Mining) [KK01] identifies all frequent patterns by a level-wise search procedure. At the first step, FSG preprocesses the input graph database and identifies all frequent single edge patterns. At step k, FSG identifies the set of frequent subgraphs that contains exactly k edges. This set is denoted as F_k. For a graph G, the *edge size* of G is the number of edges that G contains. The task at step k is subdivided into two phases: candidate subgraph proposing and candidate subgraph validation, with the detail covered below.

Candidate Subgraph Proposing. Given a set of frequent $k-1$-edge graphs F_{k-1}, FSG constructs candidate frequent k-edge subgraphs by "joining" two frequent k-1-edge subgraphs. In FSG, two graphs are "joinable" if they have the same edge size $l > 0$ and they share a common subgraph of edge size $l-1$. The "join" between two joinable graphs G_1, G_2 with edge size $k-1$ produces a set of graphs that are supergraphs of both graphs with edge size k. In other words, in FSG, the join operation is defined as:

$$FSG\_join(G_1, G_2) = \begin{cases} \{G \mid G_1 \subseteq G, G_2 \subseteq G, |E[G]| = k\} & \text{if } G_1 \text{ and } G_2 \text{ are joinable} \\ \emptyset & \text{otherwise} \end{cases}$$

We use $|E[G]|$ to denote the edge size of a graph G.

FSG applies the join operation for every pair of joinable graphs in F_{k-1} to produce a list of candidate k-edge patterns C_k. The join operation is illustrated in Figure 4.3 and the pseudo code is presented in Algorithm 2.

Example 4.4.1 *In Figure 4.3, we show the FSG join operation for a pair of graphs.*

Candidate Subgraph Validation. FSG selects the frequent subgraphs with edge size k from the set C_k by computing the support value of each graph in C_k. To compute the support value of a graph G, FSG scans the database of graphs and for each graph G' in the graph database, FSG uses subgraph isomorphism test to determine whether G is a subgraph

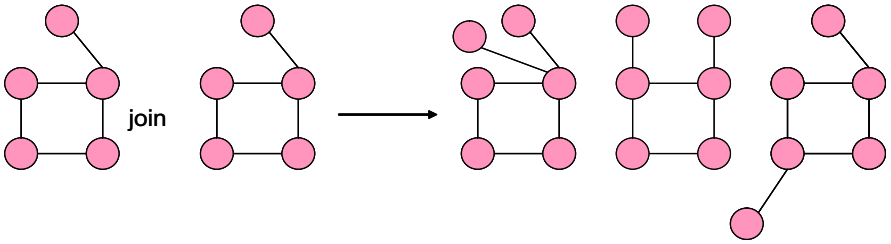

Figure 4.3: An example of applying the FSG join operation to a pair of graphs. Nodes and edges have the same label.

of G' and updates the support value of G if it is. The pseudo code of the FSG-validation is presented below in Algorithm 3.

Putting It All Together. Here we present the pseudo code for the FSG algorithm, which identifies all subgraphs F in a graph database $\mathcal{G}$ with support threshold $0 < \sigma \leq 1$. We simplified the FSG algorithm to explain its basic structure; see [KK01] for details of performance improvements.

Algorithm 1 FSG($\mathcal{G}$, σ): Frequent Subgraph Mining

1: $F_1 \leftarrow \{e \mid s(e) \geq \sigma\}$ # *all frequent edges*
2: $k \leftarrow 2$
3: **while** $F_{k-1} \neq \emptyset$ **do**
4: $\quad C_k \leftarrow$ FSG-join(F_{k-1}, k)
5: $\quad F_k \leftarrow$ FSG-validation$(C_k, \mathcal{G}, \sigma)$
6: $\quad k \leftarrow k + 1$
7: **end while**
8: $F \leftarrow \bigcup_{i \in [1,k]} F_i$

Algorithm 2 FSG-join(F_{k-1}, k): join pairs of subgraphs in F_{k-1}

1: $C_k \leftarrow \emptyset$
2: **for each** $G_1, G_2 \in F_{k-1}$ **do**
3: $\quad$ **if** there exists $e_1 \in E[G_1]$ and $e_2 \in E[G_2]$ such that $G_1 - e_1 = G_2 - e_2$ **then**
4: $\quad\quad C_k = C_k \cup \{G \mid G_1 \subset G, G_2 \subset G, \mid E(G) \mid = k\}$ # *joinable*
5: $\quad$ **end if**
6: **end for**
7: **return** C_k

Depth-First Search: The gSpan Algorithm

gSpan utilizes a depth-first algorithm to search for frequent subgraphs [YH02]. gSpan, like FSG, also preprocesses a graph database and identifies all frequent single edges at the beginning of the algorithm. gSpan designed a novel extension operation to propose candidate

Algorithm 3 FSG-validation(C_k, $\mathcal{G}$, σ): Validate Frequent Subgraphs

1: $F_k \leftarrow \emptyset$
2: **for each** $G \in C_k$ **do**
3: $s(G) \leftarrow 0$
4: **for each** $G' \in \mathcal{G}$ **do**
5: **if** $G \subseteq G'$ **then** $s(G) \leftarrow s(G) + 1$ **end** # *computing support value*
6: **end for**
7: **if** $s(G) \geq \sigma$ **then** $F_k \leftarrow F_k \cup \{G\}$ **end**
8: **end for**
9: **return** F_k

subgraphs. In order to understand this extension operation, we introduce in sequel the depth-first code representation of a graph, developed in gSpan.

Depth-First Code Of Graphs. Given a connected graph G, a depth-first search S of G produces a list of nodes in G. We denote the nodes in $V[G]$ as $1, 2, \ldots, n$ according to the order that the nodes are enumerated in S and n is the size of the graph G. We call node n as the *rightmost* node.

Each edge in G is represented by a 5-element tuple $e = (i, j, \lambda(i), \lambda(i, j), \lambda(j))$ where i, j are nodes in G ($i < j$) and λ is the labeling function of G that assigns labels to nodes and edges.

We define a total order $\preccurlyeq$ of edges in G such that $e_1 \preccurlyeq e_2$ if $i_1 < i_2$, or ($i_1 = i_2$ and $j_1 \leq j_2$).

Given a graph G and a depth-first search S, we may sort edges in G according to the total order $\preccurlyeq$ and concatenate such sorted edges together to produce a single sequence of labels. Such a sequence of labels is a *depth first code* of G. There may be many depth first codes for a graph G and the smallest one (using lexicographical order of sequences) is the *canonical depth-first search code* of G, denoted by $DFS(G)$. The depth first tree that produces the canonical form of G is its *canonical DFS tree*.

Candidate Subgraph Proposing. In gSpan, a frequent subgraph G is *extended* to a candidate frequent subgraph G' by adding one edge to G. gSpan developed very sophisticated method to carefully choose the position of the newly introduced edge to make sure that we can still enumerate all frequent subgraphs with the extension operation. See [YH02] for further detail.

Candidate Subgraph Validation. gSpan uses the same procedure used by FSG (a scan of a graph database and use subgraph isomorphism to determine the support value) to select frequent subgraphs from a set of candidates.

Comparing to level-wise search algorithm FSG, gSpan has better memory utilization due to the depth-first search and achieves an order of magnitude speedup in several benchmarks [YH02].

Putting It All Together. We present the gSpan algorithm below. We use $\sqsubseteq$ to denote a prefix relation between two strings. The procedure gSpan-validation below is the same as the FSG-validation and hence is not duplicated.

The key observation provided by gSpan is a "selective" candidate proposing strategy (line 1, Algorithm 6). By this scheme, given a graph G, gSpan only proposes subgraphs G' of G such that the DFS code of G is a prefix of that of G'. See [YH02] for proofs on how and why this selective proposing scheme works.

Algorithm 4 gSpan($\mathcal{G}$, σ): Frequent Subgraph Mining

1: $F_1 \leftarrow \{e \mid s(e) \geq \sigma\}$ *# all frequent edges*
2: $F \leftarrow F_1$
3: $k \leftarrow 1$
4: **for each** $G \in F_1$ **do**
5: $F \leftarrow F \cup$ gSpan-search$(G, k, \mathcal{G}, \sigma)$
6: **end for**

Algorithm 5 gSpan-search$(G, k, \mathcal{G}, \sigma)$

$k \leftarrow k + 1$
$C_k \leftarrow$ gSpan-extension(G, k)
$F \leftarrow F \cup$ gSpan-validation$(C_k, \mathcal{G}, \sigma)$
for each $G' \in F_k$ **do**
$F \leftarrow F \cup$ gSpan-search$(G', k, \mathcal{G}, \sigma)$
end for
return F

Algorithm 6 gSpan-extension(G, k)

1: $C_k \leftarrow \{G' \mid G \subset G', \mid E[G'] \mid = k, DFS(G) \sqsubseteq DFS(G')\}$
2: **return** C_k

Other Edge-Based Depth-First Algorithms

One performance bottle-neck of gSpan is pattern validation. This is particularly true when dealing with large and complex graphs (dense graphs with few distinct labels) since subgraph isomorphism is expensive. Instead of searching all the subgraph isomorphisms, the method proposed by Borgelt & Berhold [BB02] maintains a list of all subgraph isomorphisms ("embedding") of a frequent subgraph G to support incremental subgraph isomorphism testing.

In another edge-based depth first search method FFSM [HWP03], a hybrid candidate proposing algorithm has been invented for support both join and extension operations with improved efficiency. We cover details of FFSM in Section 5.

4.4.4 Path Based Enumeration

Below we introduce the algorithm proposed by Vanetik *et al.* that discovers all frequent subgraphs using paths as a building block [VGS02]. We name this algorithm PGM (Path-based Graph Mining).

Path Cover And Path Number Of Graphs

A *path cover* of a graph G is set of edge-disjoint paths that cover edges in G exactly once. A *minimal path cover* of a graph G is a path cover of G with the minimal number of paths. The cardinality of a minimal path cover of a graph G, denoted by $p(G)$, is the *path number* of G.

The computation of a path number is straightforward. For a connected graph $G = (V, E)$, the path number is $p(G) =\mid \{v \mid v \in E, d(v) \text{ is odd } \} \mid /2$ where $d(v)$ is the degree of a node v [VGS02].

In Figure 4.4, we show a graph G and two of its path covers $P = \{P_1, P_2\}$ and $Q = \{Q_1, Q_2\}$. Since G has four nodes with odd degree, the path number of G is $p(G) = 4/2 = 2$. Therefore both path cover P and Q are minimal path covers of G.

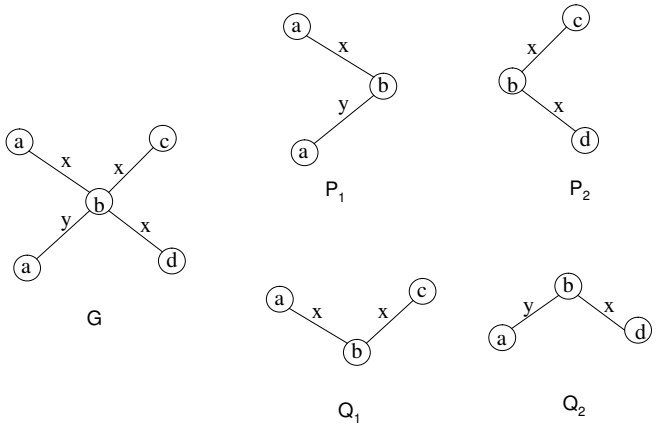

Figure 4.4: A graph G and two of its path covers.

Representing Graphs By Paths

In PGM, each graph is represented in a novel way as a set of paths and a relation among the set of paths. More specifically, PGM represents a graph G as a triple $G = (V, P, \pi)$ where

- V is the set of nodes in G,
- P is a path cover of G, and
- $\pi : \mathcal{P} \rightarrow V$ is a 1-1 mapping of nodes in path cover P to V where $\mathcal{P} = \bigcup_{p \in P} p$ is the set of all nodes in the path cover P.

The function π is named as the *composition relation* in PGM. We can prove that with a node set V, a path cover P of a graph G, and a composition relation that maps nodes in P to V, we can reconstruct the graph G exactly. The proof is given in [VGS02].

PGM Search

PGM follows the general level-wise algorithm presented in Algorithm 1 except the join operation. Below, we single out the join operation and provide details.

In PGM, each graph is represented as a set of paths P, a set of nodes V, and the composition relation of V to nodes in P. Two n-path represented graphs $G_1 = P_{1_1}, P_{1_2}, \ldots, P_{1_n}$ and $G_2 = P_{2_1}, P_{2_2}, \ldots, P_{2_n}$ are "joinable" if they differ from each other by at most one path. In other words, G_1 and G_2 are joinable if $|G_1 \cap G_2| \geq n - 1$.

For two joinable graphs G_1, G_2, PGM produces a set of graphs that are supergraphs to both G_1 and G_2. Precisely: $join(G_1, G_2)$ is defined as:

$$join(G_1, G_2) = \{G | G_1 \subset G, G_2 \subset G, p(G) = p(G_1) + 1\} \tag{4.2}$$

4.4.5 Tree Based Enumeration

Here we describe the algorithm GASTON [NK04], which introduces a frequent subgraph mining algorithm that first identifies all frequent trees and then constructs cyclic graphs. The two steps are covered in the following discussions.

Frequent Tree Identification

GASTON discovers all frequent trees using a similar strategy to that used by the edge-based depth-first algorithms. First all frequent edges are discovered. Second, single edges are extended to trees of two edges and infrequent one are deleted. The same search is repeated until no more frequent trees are identified. GASTON uses a novel tree normalization scheme that can be computed incrementally in constant time. Using this tree normalization scheme, GASTON guarantees that each frequent tree is enumerated once and only once efficiently.

Frequent Cyclic Graph Identification

For a frequent tree T, GASTON constructs a set of frequent graphs that use T as their spanning tree. Let's denote set C_E as the set of unconnected node pairs in a tree T, i.e.

$C_E = \{(i,j) \mid i < j, (i,j) \notin T\}$ (we set $i < j$ to avoid redundant pairs in an undirected tree). GASTON uses a "close" operation which introduces an edge to an pair of unconnected nodes in a tree or a graph. By applying the close operation repeatedly, GASTON enumerates all frequent cyclic graphs in which T is a spanning tree.

As a final comment for GASTON, as pointed out by Nijssen & Kok, the task of constructing frequent cyclic graphs from a tree T is similar to the frequent item set mining problem [BCG01] if we treat each edge in C_E as an "item". In fact, any algorithms that solves the frequent item set problem can potentially be adapted to solve the problem of constructing frequent cyclic graphs from a tree in GASTON.

Chapter 5

FFSM: Fast Frequent Subgraph Mining

In this chapter, we introduce an efficient algorithm for mining frequent subgraphs in graph databases: Fast Frequent Subgraph Mining (FFSM). With little extra effort, the same algorithm can be used to identify trees, cliques, quasi-cliques from a graph database, or tree patterns in a tree database [HWP03].

5.1 Introduction

5.1.1 Graph Automorphism

One of the critical problems in graph mining is the graph automorphism problem: given two graphs P and Q, determine whether P is isomorphic to Q. We solve the graph automorphism problem by assigning unique ids for graphs, or a procedure called *graph normalization*. To better explain the graph normalization procedure that is to be introduced, we introduce the following definitions.

Definition 5.1.1 *A **graph normalization function** is a 1-1 mapping ψ from G^* to an arbitrary set Γ, i.e. $\psi(G) = \psi(G') \Rightarrow G = G'$ where G^* is a graph space (i.e. all possible graphs with vertex and edge labels chosen from a fixed set).*

We work on a subclass of normalization procedures that map a graph to a sequence of labels. The label sequence $\psi(G)$ is the *canonical form* of the graph G.

5.1.2 Canonical Adjacency Matrix of Graphs

In the algorithm FFSM, we represent each graph by an adjacency matrix M such that every diagonal entry of M is filled with the label of a node and every off-diagonal entry is filled with the label of the corresponding edge, or zero if there exists no edge.

Code

There are usually more than one valid adjacency matrices for a single graph. For example, in a graph G where each node has a distinguishing label, any permutation of the node set may lead to a unique adjacency matrix, and the total number of such matrices is $n!$ where n is the number of nodes in G. As another example, at the right part of Figure 5.1, we show three adjacency matrices for the labeled graph P shown in the same Figure.

When we draw a matrix, we assume that the rows are numbered 1 through n from top to bottom, and the columns are numbered 1 through m from left to right for an $n \times m$ matrix M. For simplicity, we only show the lower triangular part of an adjacency matrix since the upper half is a mirror image of the lower one. In order to select a unique representation, we introduce the following definitions.

We define an partial order $\leqslant$ of entries in an adjacency matrix X.

Definition 5.1.2 *Given two entries $x_1 = (i_1, j_1, l_1), x_2 = (i_2, j_2, l_2)$ in X, we define $x_1 \leqslant x_2$ if*

- $i_1 < i_2$,
- $i_1 = i_2$, *and* $j_1 < j_2$

It is easy to prove that the partial order $\leqslant$ is a total order.

Definition 5.1.3 *Given an $n \times n$ adjacency matrix X of a graph G with n nodes, we define the* **code** *of X, denoted by code(X), as the sequence s formed by concatenating lower triangular entries of M (including entries on the diagonal), sorted increasingly according to $\leqslant$, or simply $s = m_{i,j}$ where $1 \leq j \leq i \leq n$.*

For an adjacency matrix M, each diagonal entry of M is referred to as a *node entry* and each off-diagonal none-zero entry is referred to as an *edge entry*. We call the *first* edge entry of M as the leftmost one in *code*(M) and the *last* edge entry as the rightmost one in *code*(M).

Example 5.1.1 *In Figure 5.1, we show three adjacency matrices for a graph P in the same Figure. For adjacency matrix M_1, the edge entry set is $\{m_{2,1}, m_{3,1}, m_{3,2}, m_{4,2}, m_{4,3}\}$ where $m_{2,1}, m_{4,3}$, and $m_{4,2}$ are the first, last, second-to-last edge entries of M, respectively.*

Canonical Form

We use standard lexicographic order on sequences to define a total order of two arbitrary codes p and q. Given a graph G, its *canonical form* is the maximal code among all its possible codes. The adjacency matrix M which produces the canonical form is the *canonical adjacency matrix* (simply CAM) of graph G, denoted by $\mathcal{M}$(G). For example, after applying the total

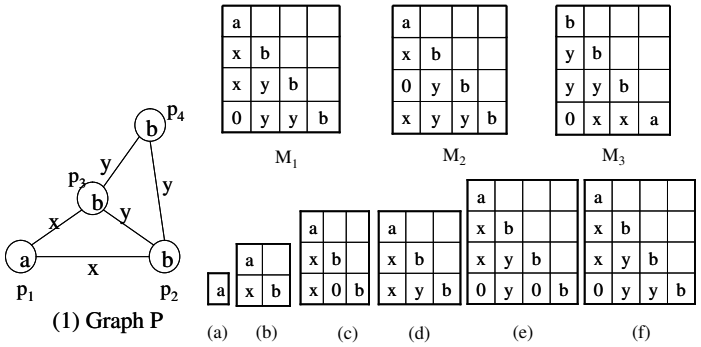

Figure 5.1: Left: A labeled graph P. Upper right: three adjacency matrices for the graph P. Lower right: examples of maximal proper submatrices. Matrix (a) is the proper maximal submatrix of matrix (b), which itself is the proper maximal submatrix of (c) and so forth.

ordering, we have $code(M_1)$ = "$axbxyb0yyb$" $\geq$ $code(M_2)$ = "$axb0ybxyyb$" $\geq$ $code(M_3)$ = "$bybyyb0xxa$". Therefore the adjacency matrix M_1 shown in Figure 5.1 is the CAM of the graph P it represents, and $code(M_1)$ is the canonical form of P.

Notice that we use maximal code rather than the minimal code used by [KK01, IWM00] in the above canonical form definition. This definition provides useful properties for subgraph mining, as explained below.

5.2 Organizing a Graph Space by a Tree

A graph space is the set of all possible graphs that draw labels from a fixed label set. In the following, we introduce a partial order on graphs and show that with the partial order we can define a tree on any graph space.

5.2.1 A Partial Order of Graphs

In order to define a partial order, we first define the maximal proper submatrix of a CAM.

Definition 5.2.1 *Given a CAM M with at least two edge entries in the last row, a matrix N is the **maximal proper submatrix** of M if N is obtained by replacing the last edge entry (and the corresponding entry of upper triangular part) of M by the value "0". Similarly, if M has only one edge entry in the last row, N is the **maximal proper submatrix** of M if N is obtained from M by removing the last row (column) of M.*

Since M represents a connected graph, it is not necessary to consider a case such that there is no edge entry in the last row of M. Several examples of the maximal proper submatrices are given at the bottom of Figure 5.1. We define that the empty matrix is the maximal proper submatrix of any matrix with size 1.

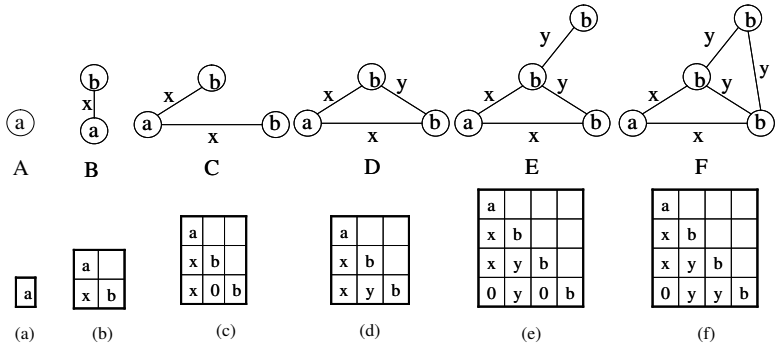

Figure 5.2: Examples of the partial order $\preccurlyeq$. Upper: A group of graphs. Lower: the related CAM representations.

Definition 5.2.2 *Given a graph space G^*, we define a binary relation $\preccurlyeq \subseteq G^* \times G^*$ such that $G \preccurlyeq G'$ if one of the following three conditions is true:*

- $G = G'$
- *$\mathcal{M}(G)$ is a maximal proper submatrix of $\mathcal{M}(G')$*
- *there exists a G'' such that $G \preccurlyeq G'' \preccurlyeq G'$*

Example 5.2.1 *In Figure 5.2,we have that $A \preccurlyeq B \preccurlyeq C \preccurlyeq D \preccurlyeq E \preccurlyeq F$ because of the maximal proper submatrix relation they have.*

Theorem 5.2.1 *$\preccurlyeq$ is a partial order*

Proof 5.2.1 *To prove that $\preccurlyeq$ is a partial order, we need to prove the following three properties:*

- *reflective: $G \preccurlyeq G$ for all graphs G*
- *anti-symmetric: $G \preccurlyeq G'$ and $G' \preccurlyeq G$ implies that $G = G'$*
- *transitive: $G \preccurlyeq G'$ and $G' \preccurlyeq G''$ imply that $G \preccurlyeq G''$.*

All the three properties are the direct results of the definition of the binary relation $\preccurlyeq$ and that of maximal proper submatrix.

5.2.2 CAM Tree

Given a graph space G^*, we define a directed graph $\mathcal{D}$ according to the partial order $\preccurlyeq$

- Each node in $\mathcal{D}$ is a distinct connected graph in G^*, represented by its CAM,
- An ordered edge (G', G) connecting two graphs G and G' if G is the minimal one such that $G' \preccurlyeq G$.

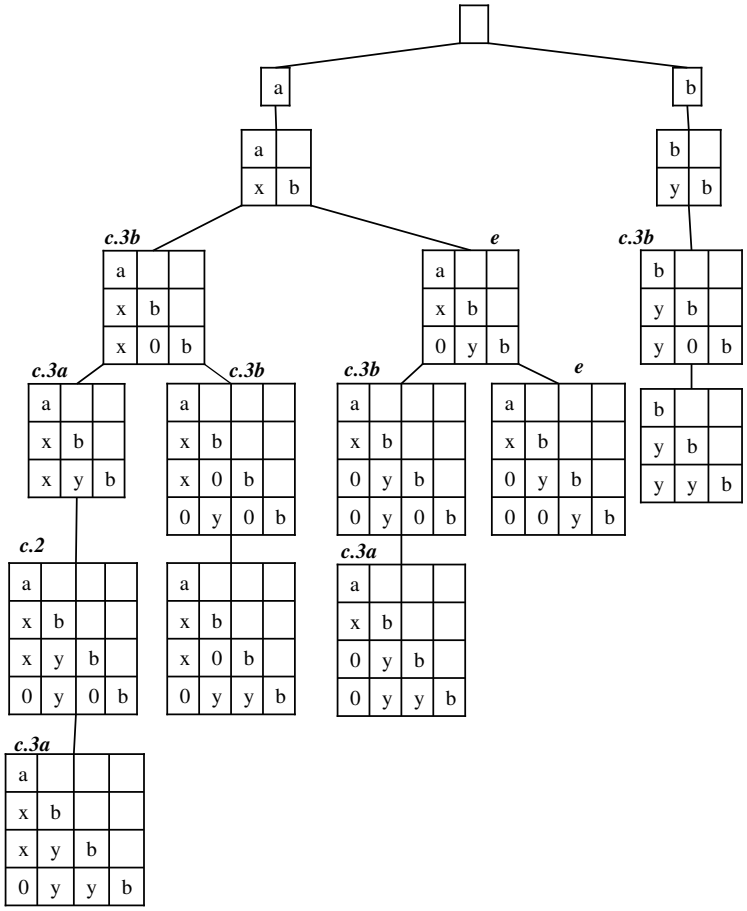

Figure 5.3: The CAM Tree of all subgraphs of the graph P in Figure 5.1. Every matrix obtained by a join operation is specified by a label starting with **c.** and then the type of the join operation e.g. **c.3a** stands for join case3a. A CAM obtained by an extension operation is labeled with **e**. The join and extension operations are discussed in Sections 5.3.1 and 5.3.2, respectively. CAMs (size ≥ 3) without label are explained in Section 5.3.3 where suboptimal CAMs are discussed. CAMs with one or no edge are obtained by a linear scanning of a graph database to obtains all frequent node and edge labels (discussed in Section 5.3.4).

We notice that each graph (with the exception of a null graph) has exactly one maximal proper submatrix. This implies that each node in $\mathcal{D}$ has at most one incoming edge. In other words, the directed graph we defined is a forest. In the following, we show that $\mathcal{D}$ is a tree, which is denoted as the *CAM tree* of the graph space. Before we do that, in Figure 5.3 we show the CAM tree of all subgraphs of the graph P from Figure 5.1.

In order to prove that the forest $\mathcal{D}$ we constructed is a rooted tree, we introduce the following theorems.

Theorem 5.2.2 *Given a CAM M of a connected graph G and M's submatrix N, N represents a connected subgraph of G.*

Proof 5.2.2 *Since N must represent a subgraph of G, it is sufficient to show the subgraph*

N represents is connected. To prove this, it is sufficient to show that in N there is no row i (with the exception of the first row) that contains no edge entry. We prove this claim by contradiction. We assume that in the matrix M, there exists at least one such row i that it does not contain any edge entry. Then we claim that we can find another row j ($j > i$) such that j contains an edge entry connecting the jth node and one of the nodes in the first $i-1$ rows (if not, the graph M corresponds to is not connected). If we perform a swap of row i and j, we claim that the code of the newly obtained adjacency matrix is lexicographically greater than that of M. This fact contradicts to the definition of CAM, which asserts the CAM of a graph has the largest code.

With the previous proved theorem, we have the following theorem.

Theorem 5.2.3 *The graph $\mathcal{D}$ we constructed in Section 5.2.2 is a rooted tree with the empty graph as its root.*

Proof 5.2.3 *To prove that a forest is a tree, we only need to show that it is connected. Specifically for the graph $\mathcal{D}$, we need to show that for all nodes G, there exists one sequence of nodes $G_1, G_2, \ldots, G_n$ such that G_1 is the root, $G_n = G$ and G_i is connected to G_{i+1} for $1 \leq i < n$. This is a direct results that the previous proved Theorem 5.2.2*

5.3 Exploring the CAM Tree

The current methods for enumerating all the subgraphs might be classified into two categories: one is the join operation adopted by FSG and AGM [IWM00, KK01]. A join operation takes two "joinable" frequent k-edge graphs G_1 and G_2 and produces a $(k+1)$-edge graph candidate G such that both G_1 and G_2 are subgraphs of G. Two k-edge graphs are *joinable* if they share a common $(k-1)$-edge subgraphs. The join operation is expensive, as shown in [KK01], in that a single join operation might generate many graph candidates and one candidate might be redundantly proposed by many distinct join operations.

On the other hand, [BB02, YH02] use an extension operation to grow a frequent graph. An extension operation produces a $(k+1)$-edge graph candidate from a frequent k-edge graph G by adding one additional edge to G (with or without introducing an additional node). This operation is also costly since for a given graph, there are many nodes in the graph that an additional edge might be attached to.

In order to derive a hybrid method with improved efficiency, we list some of the key challenges to achieve:

- Can we interleave join and extension operation to achieve maximal efficiency?
- Can we design a join operation such that every distinct CAM is generated only once?

- Can we improve a join operation such that only a few graphs can be generated from a single operation (say at most two)?

- Can we design an extension operation such that all the edges might be attached to only a single node rather than many nodes in a graph?

In order to meet these challenges, we have introduced two new operations, FFSM-Join and FFSM-Extension, have augmented the CAM tree with a set of *suboptimal canonical adjacency matrices*, and have designed an embedding based subgraph enumeration method. Experimental evidence demonstrates that our method can achieve an order of magnitude speed up over the current state-of-the-art subgraph mining algorithm gSpan [YH02]. Below we discuss FFSM-join and FFSM-extension.

5.3.1 FFSM-Join

The purpose of the join operation is "superimposing" two graphs to generate a new candidate graph. Depending on the different characteristics of the graphs, the join operation in our algorithm might produce one or two graph candidates.

Given an adjacency matrix A of a graph G, we define A as an "*inner*" matrix if A has at least two edge entries in the last row. Otherwise, A is an "*outer*" matrix. Given two adjacency matrices A $(m \times m)$ and B $(n \times n)$ sharing the same maximal proper submatrix, A's last edge $a_{m,f}$, and B's last edge $b_{n,k}$, we define *join(A, B)* by the following three cases:

join case 1: both A and B are inner matrices $(m = n$ in this case)

1: **if** $f \neq k$ **then**

2: $join(A, B) = \{C\}$ where C is a $m \times m$ matrix such that

$$c_{i,j} = \begin{cases} b_{i,j} & i = n, j = k \\ a_{i,j} & \text{otherwise} \end{cases}$$

3: **else**

4: $join(A, B) = \emptyset$

5: **end if**

join case 2: A is an inner matrix and B is an outer matrix $(m = n - 1)$

1: $join(A, B) = \{C\}$ where C is a $n \times n$ matrix and

2:

$$c_{i,j} = \begin{cases} a_{i,j} & 0 < i, j \leq m \\ b_{i,j} & \text{otherwise} \end{cases}$$

join case 3: both A and B are outer matrices $(m = n)$

1: let matrix D be a $(m+1) \times (m+1)$ matrix where (case 3b)

$$d_{i,j} = \begin{cases} a_{i,j} & 0 < i, j \leq m \\ b_{m,j} & i = m+1, 0 < j < m \\ 0 & i = m+1, j = m \\ b_{m,m} & i = m+1, j = m+1 \end{cases}$$

2: **if** $(f \neq k, a_{m,m} = b_{m,m})$ **then**

3: C is $m \times m$ matrix where (case 3a)

$$c_{i,j} = \begin{cases} b_{i,j} & i = n, j = k \\ a_{i,j} & \text{otherwise} \end{cases}$$

4: $join(A, B) = \{C, D\}$

5: **else**

6: $join(A, B) = \{D\}$

7: **end if**

In join case 3, when joining two outer matrices M_1 and M_2 (both with size m), we might obtain a matrix with the same size. We refer this join operation as *case3a*. It is also possible that we obtain a matrix having size $(m+1)$ and this case is referred as *case3b*.

We notice that the join operation is symmetric with respect to A and B with the only exception of join case 3b. In other words, $join(A, B) = join(B, A)$ for join case 1, 2 and 3a and $join(A, B) \neq join(B, A)$ in join case3b. In order to remove the potential duplications resulting from this symmetry, we require that $code(A) \geq code(B)$ in all join cases except join case 3b. Equality is permitted since self-join is a valid operation. If the inequality is not satisfied ($code(A) < code(B)$), a join operation produces an empty set.

Figure 5.4 shows examples for the join operation for all four cases. At the bottom of Figure 5.4, we show a case where a graph might be redundantly proposed by FSG $\binom{6}{2} = 15$ times (joining of any pair of distinct five-edge subgraphs G_1, G_2 of the graph G will restore G by the join operation proposed by FSG). As shown in the graph, FFSM-Join completely removes the redundancy after "sorting" the subgraphs by their canonical form.

However, the join operation is not "complete" in the sense that it may not enumerate all the subgraphs in the CAM tree. Interested readers might find such examples in the CAM tree we presented in Figure 5.3. Clearly we need another operation, which is discussed below.

5.3.2 FFSM-Extension

Another enumeration technique in the current subgraph mining algorithms is the extension operation that proposes a k+1-edge graph candidate G from a k-edge graph G_1 by introducing one additional edge. In these algorithms, the newly introduced edge might connect two

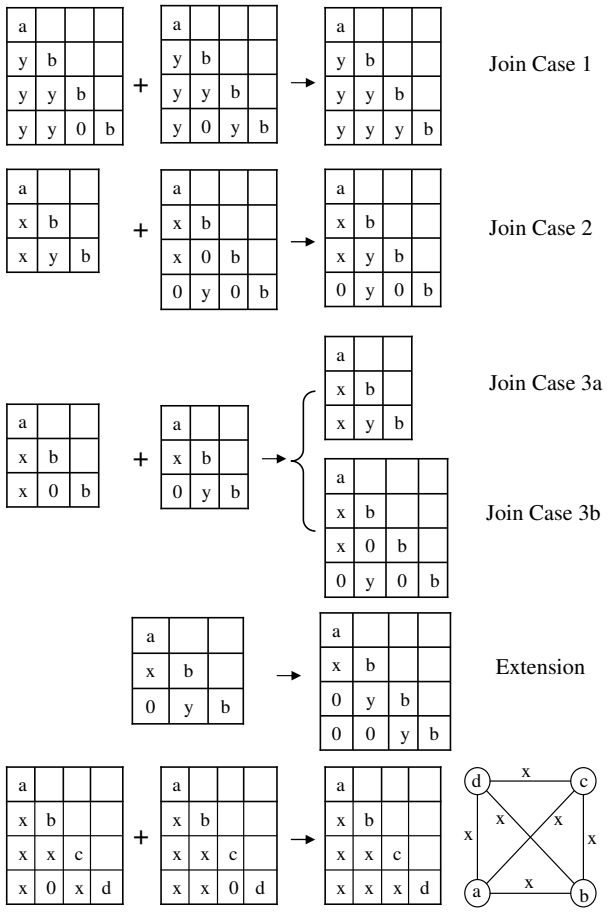

Figure 5.4: Examples of the join/extension operation

existing nodes or connect an existing node and a node introduced together with the edge. A simple way to perform the extension operation is to introduce every possible edge to every node in a graph G. This method clearly has complexity of $O(\Sigma_V \times \Sigma_E \times |G|)$ where Σ_V, Σ_E stand for the set of available vertex and edge labels for a graph G, respectively for a single extension. It suffers from the large size of graph candidates as well as the large amount of available node/edge labels.

gSpan [YH02] developed an efficient way to reduce the total number of nodes that need to be considered. In gSpan, the extension operation is only performed on nodes on the "rightmost path" of a graph. Given a graph G and one of its depth first search trees T, the *rightmost path* of G with respect to T is the rightmost path of the tree T. gSpan chooses only one depth first search tree T that produces the canonical form of G for extension. See [YH02] for further details about the extension operation.

In FFSM, we further improve the efficiency of the extension operation by choosing only a single node in a CAM and attaching an newly introduced edge to it together with an additional node. As proved by Theorem 5.3.1, this extension operation, combined with the

join operation, unambiguously enumerates all the nodes in the CAM tree.

In the following pseudo code, we present the extension operation.

Algorithm 7 FFSM-Extension(A)

1: **if** (A is an outer adjacency matrix) **then**
2: **for** $(n_l, e_l) \in \Sigma_V \times \Sigma_E$ **do**
3: $S \leftarrow \emptyset$
4: create an $n \times n$ matrix $B = (b_{i,j})$ such that
5:

$$b_{i,j} = \begin{cases} a_{i,j} & 0 < i, j \leq n \\ 0 & i = n+1, 0 < j < n \\ e_l & i = n+1, j = n \\ n_l & i = n+1, j = n+1 \end{cases}$$

6: $S \leftarrow S \cup \{B\}$
7: **end for**
8: **else**
9: $S \leftarrow \Phi$
10: **end if**

5.3.3 Suboptimal CAM Tree

Using the CAM tree of the graph P in Figure 5.3, we can verify that the join and extension operations, even combined together, can not enumerate all subgraphs in P. We investigated this and found this problem can be solved by introducing the suboptimal canonical adjacency matrices, as defined below.

Definition 5.3.1 *Given a graph G, a **suboptimal Canonical Adjacency Matrix** (simply, suboptimal CAM) of G is an adjacency matrix M of G such that its maximal proper submatrix N is the CAM of the graph that N represents.*

By definition, every CAM is a suboptimal CAM. We denote a *proper suboptimal CAM* as a suboptimal CAM that is not the CAM of the graph it represents. Several suboptimal CAMs (the matrices with dotted boundaries) are shown in Figure 5.5. Clearly, all the suboptimal CAMs of a graph G could be organized in a tree in a similar way to the construction of the CAM tree. One such example for the graph P in Figure 5.1 is shown in Figure 5.5.

With the notion of suboptimal CAM, the suboptimal CAM tree is "complete" in the sense that all vertices in a suboptimal CAM tree can be enumerated using join and extension operations. This is formally stated in the following theorem.

Theorem 5.3.1 *() For a graph G, let $C_{k-1}(C_k)$ be set of the suboptimal CAMs of all the $(k-1)$-vertex (k-vertex) subgraphs of G ($k \geq 3$). Every member of set C_k can be enumerated unambiguously either by joining two members of set C_{k-1} or by extending a member in C_{k-1}.*

Proof 5.3.1 *Let A be a $m \times m$ suboptimal CAM in set C_k. We consider the following five cases according to the edge entries in A's last row and second-to-last row:*

- *TypeA M has three or more edge entries in the last row;*
- *TypeB M has exactly two edge entries in the last row;*
- *TypeC M has exactly one edge entry in the last row and more than one edge entries in the second-to-last row;*
- *TypeD M has exactly one edge entry $e_{m,n}$ in the last row and one edge entry in the second-to-last row and $n \neq m-1$;*
- *TypeE M has exactly one edge entry $e_{m,n}$ in the last row and one edge entry in the second-to-last row and $n = m-1$;*

As shown in the Appendix in [HWP03], a TypeA suboptimal CAM can be produced by two suboptimal CAMs following join case1. Similarly, a TypeB suboptimal CAM corresponds to the join case3a, a TypeC suboptimal CAM corresponds to join case2, a TypeD suboptimal CAM corresponds to join case3b, and a TypeE suboptimal CAM corresponds to the extension operation.

5.3.4 Mining Frequent Subgraphs

In the previous discussions, we present a novel data structure (CAM tree) for organizing all connected subgraphs of a single connected undirected graph. This scheme can be easily extended to a set of graphs (connected or not), denoted as a graph database, and a single CAM tree can be built for such a graph database. If we have such a tree built in advance (regardless of the required space and computational complexity), a traversal of the tree can easily reveal the set of distinct subgraphs of the graph database. For each such subgraph, its support can be determined by a linear scan of the graph database, frequent ones can be reported subsequently. This method clearly suffers from the huge number of available subgraphs in a graph database and therefore is unlikely scale to large graph databases.

In the following pseudo code, we present an algorithm which takes advantage of the following simple fact: if a subgraph G is not frequent (i.e. the support of G is less than a user posted threshold), none of its supergraphs is frequent. This suggests that we can stop building a branch of the tree as soon as we find that the current node does not have sufficient support in a graph database.

In the pseudo code shown in Algorithm 8, symbol $\mathcal{M}(G)$ denotes the CAM of a graph G. $X.isCAM$ is a boolean variable indicate whether the matrix X is the CAM of the graph it represents. $s(G)$ is the support value of a graph G (or its CAM $\mathcal{M}(G)$).

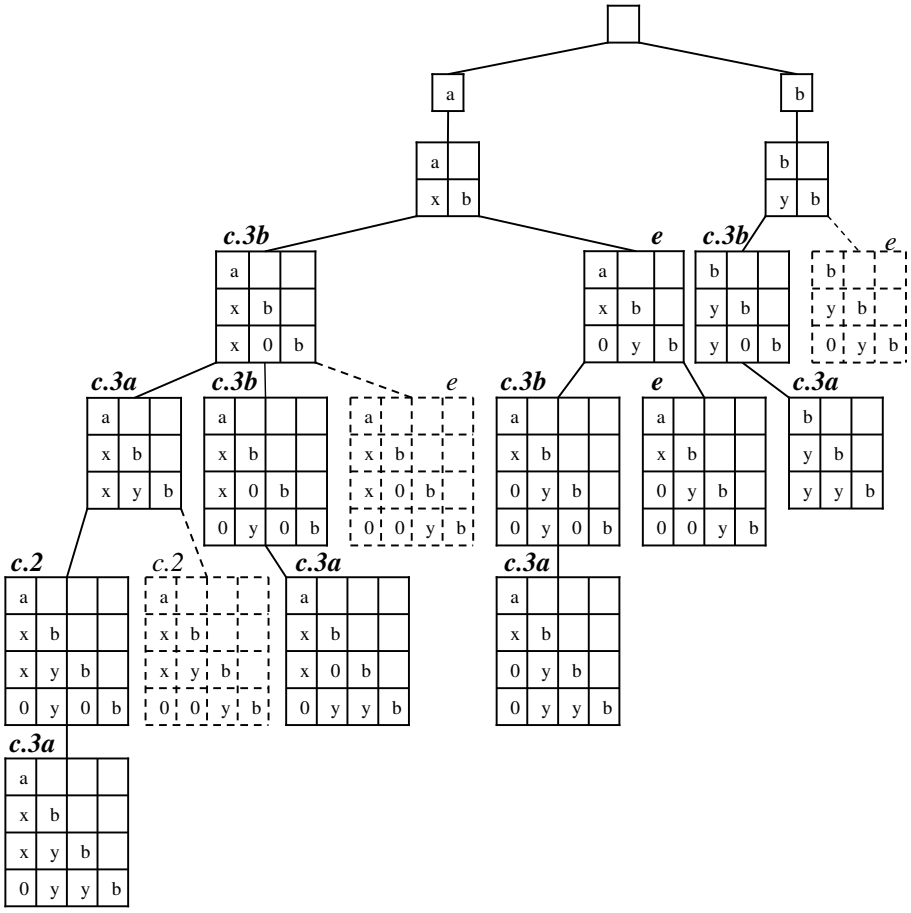

Figure 5.5: The Suboptimal CAM Tree for the graph P that is shown in Figure 5.1. Matrices with solid boundary are CAMs and those with dashed line boundary are proper suboptimal CAMs. The label on top of an adjacency matrix M indicates the operation by which M might be proposed from its parent. The labeling follows the same conventions used in Figure 5.3.

Algorithm 8 FFSM($\mathcal{G}$, σ)

1: $P \leftarrow \{\mathcal{M}(e) \mid e \text{ is an edge}, s(e) \geq \sigma\}$
2: $F \leftarrow$ FFSM-Explore(P, P)
3: **return** F

5.3.5 Performance Optimizations

One of the key challenges for efficient subgraph mining is support value computation. A brute force way to compute the support value of a subgraph pattern P is to perform a linear scan of a database. For each graph G in the database, we use subgraph matching to determine whether a graph pattern P occurs in G. Subgraph matching is known to be NP-complete and therefore we want to avoid subgraph matching as much as possible in frequent subgraph mining.

From a practical point of view, the above scheme can be significantly sped up by recording

Algorithm 9 FFSM-explore(P, F)

```
1: for each X ∈ P do
2:   if (X.isCAM) then
3:     F ← F ∪ {X}, C ← ∅
4:     for each Y ∈ P do
5:       C ← C ∪ FFSM-Join(X, Y)
6:     end for
7:     C ← C ∪ FFSM-Extension(X)
8:     C ← {G|G ∈ C, G is frequent, G is suboptimal }
9:     F ← F∪ FFSM-Explore(C, F)
10:  end if
11: end for
12: return F
```

the previous matched vertices (denoted as an *embedding*) and basing the subsequent search on the previous recorded information. To further explain this point, we define an embedding as follows:

Definition 5.3.2 *Given an arbitrary $n \times n$ adjacency matrix A and a labeled graph $G = (V, E, \Sigma_V, \Sigma_E, l)$, a vertex list $L = u_1, u_2, \ldots, u_n \subset V$ is an **embedding** of A in G if:*

- *(Condition I:)* $\forall\ i \in [1, n], (a_{i,i} = l(u_i))$,
- *(Condition II:)*$\forall\ (i, j) \in [1, n] \times [1, n], (a_{i,j} \neq 0 \Rightarrow a_{i,j} = l(u_i, u_j))$

The set of all embeddings of a matrix in a database is defined as its *embedding set*. Given two suboptimal CAMs and their embedding sets, we could modify the join and extension operations that we presented before to let them not only propose candidates but also calculate candidates' embedding sets.

Let's first study the join case 1, for which claim the following fact.

Theorem 5.3.2 *Given three suboptimal CAMs P, Q, A and A = FFSM-joinCase1(P, Q), we have $O_A = O_P \cap O_Q$, where O_P and O_Q are the embedding sets of suboptimal CAM P and Q, respectively.*

Proof 5.3.2 *Consider an embedding L of the matrix A in a graph G, clearly L is an embedding of the matrix P. This is because (1) condition I is automatically satisfied and (2) Condition(II) must also hold since A contains all the non-zero entries of both P and Q.*

On the other hand, if we have an list of nodes L which is an embedding of both P and Q, the same list must be a embedding of A (noticing A contains no more non-zero entries than those found in either P or Q).

Similarly, for join case 2, ($\{A\} = join(P, Q)$, P is an inner matrix and Q is an outer matrix), we have $O_A = \{L \mid L = u_1, u_2, \ldots, u_{n-2}, u_{n-1}, u_n, L \in O_q, L' = u_1, u_2, \ldots, u_{n-2}, u_{n-1} \in O_P\}$. For join case 3a ($\{A\} = join(P, Q)$, both P and Q are outer matrices, A has the same size as P and Q after join), we have $O_A = O_P \cap O_Q$. For join case 3b ($\{A\} = join(P, Q)$, both P and Q are outer matrices, and A has size one greater than that of P and Q after join), we have $O_A = \{L \mid L = u_1, u_2, \ldots, u_n, L' = u_1, u_2, \ldots, u_{n-3}, u_{n-2}, u_{n-1} \in O_p, L'' = u_1, u_2, \ldots, u_{n-3}, u_{n-2}, u_n \in O_q\}$.

For the extension operation, we present the embedding set calculation as a pseudo code below:

FFSM-Extension-Embedding

input: A suboptimal $n \times n$ CAM A
output: A set S of all suboptimal CAMs extended from A and their embedding sets

1: **for** embedding $L = u_1, u_2, \ldots, u_n \in O_A$ **do**
2: let G be the graph contains vertices in L
3: **for** node $v \in V[G] \wedge v \notin L \wedge (v, u_n) \in E[G]$ **do**
4: create a $(n+1) \times (n+1)$ Matrix B where

$$b_{i,j} = \begin{cases} a_{i,j} & 0 < i, j \leq n \\ 0 & i = n+1, 0 < j < n \\ l_G(v, u_n) & i = n+1, j = n \\ l_G(v) & i = n+1, j = n+1 \end{cases}$$

5: $O_B \leftarrow O_B \cup \{(g_i, Lv)\}$
6: $S \leftarrow S \cup \{B\}$
7: **end for**
8: **end for**

where l_G is the mapping used by G to map the vertices and edges to their labels. Lv is the list concatenated by list L and a single node v.

5.4 Performance Comparison of FFSM

We have evaluated the performance of the FFSM algorithm with various types of graphs. The experimental study was carried out using a single processor of a 2GHz Pentium PC with 2GB memory, running RedHat Linux 7.3. The FFSM algorithm was implemented using the C++ programming language and compiled using g++ with O3 optimization. We compared our algorithm to gSpan, which is the state-of-the-art algorithm for graph mining. The gSpan executable, compiled in a similar environment, was provided by X. Yan and J. Han [YH02].

5.4.1 Synthetic Data Sets

Data Sets

We used a graph generator offered by M. Kuramochi and G. Karypis [KK01] to generate synthetic graph databases with different characteristics. There are six parameters to control the set of synthetic graphs:

- $| D |$, total graph transactions generated,
- $| T |$, average graph size for the generated graphs, in terms of number of edges,
- $| L |$, the total number of the potentially frequent subgraphs,
- $| I |$, the size of the potentially frequent subgraphs, in terms of number of edges,
- $| V |$, total number of available labels for vertices, and
- $| E |$, total number of available labels for edges.

We use a single string to describe the parameter settings, e.g. "$D10kT20L200I9V4E4$" represents a synthetic graph database which contains a total of $|D| = 10$k (10000) graph transactions. Each graph on average contains $|T| = 20$ edges with up to $|V| = 4$ vertex labels and $|E| = 4$ edge labels. There are total of $|L| = 200$ potential frequent patterns in the database with average size $|I| = 9$.

Performance Comparison

In Figure 5.6, we show how the FFSM algorithm scales with increasing support. The total number of identified frequent subgraphs is also given.

At the left in Figure 5.7, we show performance comparison between FFSM and gSpan with different average graph sizes (left) or different number of node/edge labels (right). On almost all circumstances, FFSM is faster than gSpan, though the value of the speedup varies from data set to data set.

5.4.2 Chemical Compound Data Sets

Data Sets

We use three chemical compound data sets to evaluate the performance of the FFSM algorithm. The first data set is the PTE data set [SKMS97] that can be downloaded from http::/web.comlab.ox.ac.uk/oucl/research/areas/machlearn/PTE/. This data set contains 337 chemical compounds each of which is modeled by an undirected graph. There are a total of 66 atom types and four bond types (single, double, triple, aromatic bond) in the data

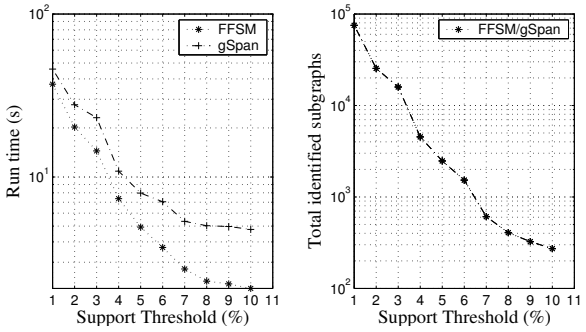

Figure 5.6: FFSM and gSpan performance comparison under different support values. Parameters used: D10kT20I9L200E4V4.

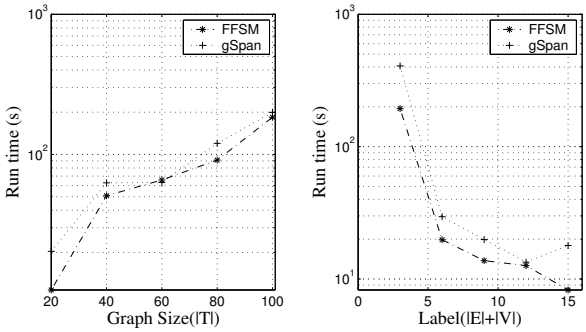

Figure 5.7: FFSM and gSpan performance comparison under different graph sizes ($|T|$) ranging from 20 to 100 (left) or different total labels ($|V| + |E|$) ranging from 3 to 18 (right). The ratio of the $|V|$ to $|E|$ is fixed to 2:1 for any given total number of labels. For example, if there are total 15 labels, we have 10 vertices label and 5 edge label. Other parameters setting: D10kI7L200E4V4 (left) and D10kT20I7L200 (right). The support threshold is fixed at 1% in both cases

set. The atoms and bonds information are stored in two separate files and we follow exactly the same procedure described in [YH02] to construct the graph representations of chemical structures.

The next two data sets are derived from the DTP AIDS Antiviral Screen data set from National Cancer Institute. Chemicals in the data set are classified into three classes: confirmed active (**CA**), confirmed moderately active (**CM**) and confirmed inactive (**CI**) according to experimentally determined activities against HIV virus. There are a total of 423, 1083, and 42115 chemicals in the three classes, respectively. For our own purposes, we formed two data sets consisting of all CA compounds and of all CM compounds and refer to them as DTP CA and DTP CM respetively. The DTP datasets can be downloaded from http://dtp.nci.nih.gov/docs/aids/aids_data.html.

Performance Comparison

We evaluate the performance of FFSM using various support thresholds. The result is summarized in Figure 5.8 and 5.9. We find that FFSM has a maximal 7 fold speedup over gSpan on the DTP CM data set. For DTP CA and PTE data set, FFSM usually has a 2 to 3 fold speedup from gSpan.

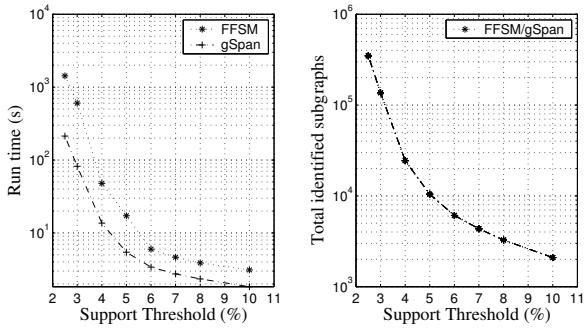

Figure 5.8: Left: performance comparison of FFSM and gSpan with different support values for the DTP CM data set. Right: the total number of frequent patterns identified by the algorithms.

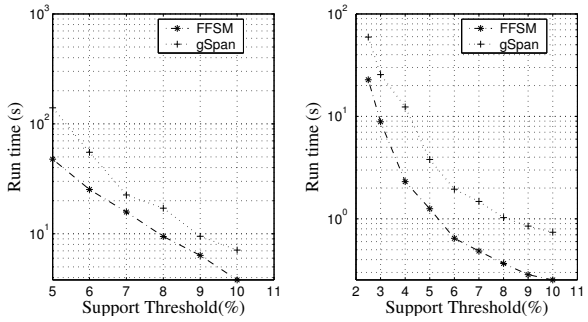

Figure 5.9: performance comparison of FFSM and gSpan with different support values for the DTP CA data set (left) and PTE (right).

σ	FFSM(s)	FSG(s)
100	0.0433	0.433
95	0.2	1.633
90	0.537	3.6
85	2.243	14.1
80	11.64	61.433
75	104.58	700.217
70	1515.15	17643.667

Table 5.1: Performance comparison between FFSM and FSG. σ support threshold (percentage). Performance of FFSM and FSG are measured in seconds.

5.4.3 Protein Contact Maps

Data Sets

We collect a group of serine proteases from the Structure Classification of Proteins database [MBHC95] with SCOP id 50514 (eukaryotic serine proteases). For each protein, we map it to a graph, known as the "contact map" of the protein, in the following way:

- A node represents an amino acid residue in a protein, labeled by the residue identity
- An edge connects two residues as long as the two residue are in "contact". Edges are not labeled.

In our representation, an amino acid residue is abstracted as two element tuple (p, id) where p is a point representing the C_α atom of the residue and id is the identity of the residue. Given a set of points in a 3D space (each point represents a C_α atom in a protein), we compute all possible delaunay tessellations of the point set (in the format of point pairs), with the condition that each point may move away from its location by up to $\epsilon > 0$ Å. The result is known as the almost-Delaunay edges for the point set [BS04]. We define that two residues are in *contact* if they are connected by an almost-Delaunay edges with $\epsilon = 0.1$ Å and with length up to 8.5 Å. The same data set and the way we represent proteins as graphs are discussed in detail in [HWB+04] and the data set is downloadable from http://www.cs.unc.edu/~huan/FFSM.shtml.

Performance Comparison

The current gSpan is specifically developed for small graphs (with no more than 200 edges in any graphs in a data set). We compare FFSM with another graph mining algorithm FSG [KK01]. FFSM always an order of magnitude faster than FSG. Table 5.1 summarizes the results.

We showed the performance comparison between different graph mining algorithms in this chapter. In the next a few chapters, we are going to show how graph mining may be applied to protein structures to derive common structure patterns. Before we do that, we first study two ways to further improve the performance of the current subgraph mining algorithm.

Chapter 6

SPIN: Tree-Based Frequent Subgraph Mining

6.1 Introduction

In this chapter we present an algorithm to find a subset of recurring subgraphs that they are not part of any other frequent ones, or *maximal frequent subgraphs*.

Our work is motivated by the effort to improve the computational efficiency of frequent subgraph mining when dealing with large graph databases. One important issue here is handling the huge number of recurring patterns. The phenomenon is well understood in mining frequent itemsets. Given a frequent itemset I, any subset of I is also frequent by common support definition [AS94] and hence the total number of frequent itemsets grows exponentially in the number of items contained in an itemset. Mining graphs have the same problem: any subgraph of a frequent graph is frequent and the total number of frequent subgraphs can grow exponentially. Practically we do observe a sharp increase of the total number of subgraphs when we mine even moderate sized graph databases [HWP03].

In this chapter, we propose a graph mining algorithm that mines only maximal frequent subgraphs, i.e. graphs whose supergraphs are all infrequent. Mining maximal frequent subgraphs offers several advantages in processing large databases.

- Maximal subgraph mining significantly reduces the total number of mined subgraphs. In experiments on chemical data sets, the total number of frequent subgraphs is up to one thousand times greater than the number of maximal subgraphs. We can save both space and subsequent analysis effort if the number of mined subgraphs is significantly reduced.

- Several "pruning" techniques, which are detailed in this chapter, can be efficiently integrated into the mining process and dramatically improve the performance of the mining algorithm.

- We do not loose information since the non-maximal frequent subgraphs can be reconstructed from the maximal subgraphs reported. To get the actual frequency (support) of non-maximal subgraphs requires examination of the original database, but it is certain to be at least as high as the frequency of the maximal subgraph. In addition, the techniques used in [PDZH02] can be easily adapted to approximate the support of all frequent subgraphs within some error bound.

- In some applications such as discovering structure motifs in a group of homology proteins [HSS+02, HWW+04], maximal frequent subgraphs are of interest since they encode the maximal structure commonalities within the group.

In this chapter we show that maximal subgraph mining can be efficiently performed. Our mining method is based on a graph mining framework in which we first mine all frequent tree patterns from a graph database and then construct maximal frequent subgraphs from trees. This approach offers asymptotic advantages compared to using subgraphs as building blocks, since tree normalization is a simpler problem than graph normalization. The proposed method enables us to integrate well-developed techniques from mining maximal itemsets and knowledge gained in graph mining into a new algorithm. According to our experimental study, such a combination can offer significant performance speedup in both synthetic and real data sets. The framework of our method is versatile. Depending on the particular tree mining algorithm, the search can be either breadth-first or depth-first (preferred due to its better memory utilization). It can also be designed to mine all frequent subgraphs without major modifications.

In summary, we make three contributions in this chapter:

- we propose a novel algorithm SPIN (**SP**anning tree based maximal graph m**IN**ing) to mine maximal frequent subgraphs of large graph databases,

- we integrate several optimization techniques to speed up the mining process,

- we perform an extensive analysis of the proposed algorithm on graph databases with different characteristics.

The remainder of the chapter is organized as follows. Section 2 presents a formal description of the maximal frequent subgraph mining problem. Section 3 presents the data structure and the proposed search algorithm. Section 4 presents the results of our experimental study using synthetic graph databases and two benchmark chemical data sets. We conclude the chapter with a discussion, related works, and conclusion.

6.2 Background

We start this section defining maximal frequent subgraph. Notations related to trees are introduced because of the close relation of these notations to this chapter. All graphs (including trees) discussed here are single-labeled undirected graphs.

6.2.1 Maximal Frequent Subgraph Mining

We define maximal frequent subgraphs below:

Definition 6.2.1 *Given a set $\mathcal{G}$ of labeled graphs and a threshold $0 < \sigma \leq 1$, a subgraph G is a maximal frequent subgraph if:*

- $s(G) \geq \sigma$, *and*
- $\nexists G'$, *($G \subset G'$ and $s(G') \geq \sigma$).*

Example 6.2.1 *In Figure 6.1 we show three labeled graphs P, Q and S. Both Q and S are subgraphs of P and the mapping f: $q_1 \rightarrow p_1, q_2 \rightarrow p_2, q_3 \rightarrow p_3$ represents an subgraph isomorphism from graph Q to P as defined in Chapter 4. The graph Q has support value 2/3 in the set $\{P, Q, S\}$ because Q is a subgraph of P and Q, but not a subgraph of S. Given a threshold $\sigma = 2/3$, both graphs Q and S are maximal frequent subgraphs in the data set $\{P, Q, S\}$ since no proper supergraph of either of them is frequent.*

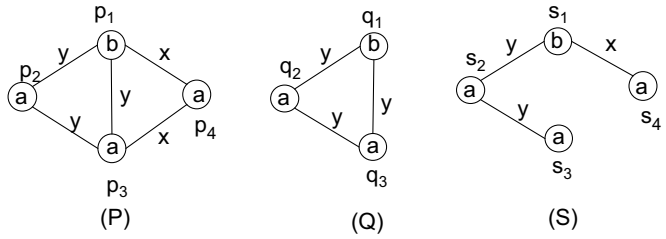

Figure 6.1: Example of a graph database $\mathcal{G}$ of three labeled graphs with subgraph relation. We assume that the node and edge labels are ordered alphabetically.

The **maximal frequent subgraph mining problem** is given a set $\mathcal{G}$ of labeled graphs and a threshold $0 < \sigma \leq 1$, to identify all maximal frequent subgraphs G such that $s(G) \geq \sigma$.

6.2.2 Labeled Trees

We also use the following tree related notations.

- A *(free) tree* is a connected acyclic graph.

- A *subtree* of a free tree T is a subgraph of T, which implies that every subtree itself is a free tree.

- A *rooted tree* is a free tree with a node designated as its *root*.

- In a rooted tree, a node u on the path from the root to a node v is an *ancestor* of v.

- If u is v's connected ancestor, u is v's parent and consequently v is u's child.

- A group of nodes sharing the same parent are *siblings*. A node is an *internal node* if it has at least one child; otherwise, it is a *leaf*.

- An *ordered rooted tree* is a rooted tree where the children of each internal node are ordered i.e. we can refer the children of a node as its first child, second child, and so on so forth.

- Tree may have labels, i.e. each node and edge has an associated label. In a labeled tree, a labeling function δ is provided as a mapping from a tree's node/edge to its associated label. We use the same symbol (δ) for both trees and graphs and the difference is usually understood from the context.

- Given an ordered rooted tree T, we define the *right-most leaf* as the leaf x which all of its ancestors (including x itself) are ordered the least among their siblings; the ordered rooted tree T_1 obtained by remove the right-most leaf from T is defined as the *left-most* subtree of T.

Example 6.2.2 *We present three labeled, ordered, and rooted trees in Figure 6.2. Throughout the paper, we assume that root of a ordered rooted tree is shown on top and the children of each internal node are ordered left to right. Node t_6 of tree T is its right-most leaf.*

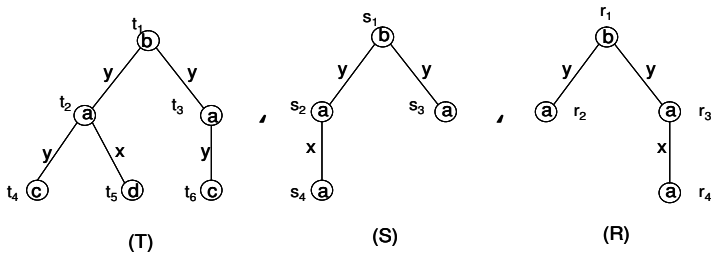

Figure 6.2: Example of three labeled ordered rooted trees and the order $\succeq$ among such trees.

6.2.3 Normalizing Labeled Ordered Rooted Trees

In the following discussion, we define a string representation of a tree. We introduce two special symbols "$", "#" which we assume no tree may have them as labels. We further assume all labels of trees are ordered alphabetically and are greater than "$", which is greater than "#". For every labeled, ordered, and rooted tree T, there is a unique string presentation as suggested by Chi *et al.* [CYM03]. The way to derive the string representation can be broken into two steps: preprocessing and label derivation. We discuss the two steps below.

Preprocesssing

In the preprocessing step, for each edge (u, v) in T (and assume v is u's child), we create an artificial label for v as a two-element sequence $e_l v_l$, where e_l and v_l are the labels of the edge (u, v) and the node v, respectively[1].

Label Derivation

In the label derivation step, we perform a breadth-first enumeration of the nodes in the tree, starting from the root of the tree and following the order among children when visiting sibling nodes. The string obtained by concatenating the artificial labels of the nodes according to the traversal order, using "$" to group siblings and "#" to label the end of the string, is the string representation of the tree.

Example 6.2.3 *The tree T shown in Figure 6.2 has the string representation b$yaya$ycxdyc#. In this example, we start with the label of the root, b. By breadth-first enumeration, we concatenate yaya to b since it is the artificial label of the first and the second children of the root of T. We add $ to the partial-completed string to group the two siblings. We continue the same enumeration for the rest of the nodes in T; add a # to label the end of the representation; and finally obtain the string b$yaya$ycxdyc# as T's string representation.*

We use string's lexicographical order to define a total order $\succeq$ of all labeled, ordered, and rooted tree i.e. $T \succeq S$ if the string representation of T is lexicographically greater or equal to that of S. For example, for the trees shown in Figure 6.2, we can easily verify that $T \succeq S \succeq R$.

6.2.4 Normalizing Free Trees

A free tree T can be transformed to an ordered rooted tree by randomly designating a root and arbitrarily imposing order of children for every internal node. Each such tree is referred to as the *rooted ordered version* of T. Given a free tree T, T's *canonical representation*, denoted by

[1] Whether an edge label precedes a node label or not is not important. We just use one order throughout the chapter.

$\mathcal{T}(T)$, is the maximal ordered rooted tree among all T's ordered rooted versions; the process to find such canonical representation is referred to as the *tree normalization* procedure and details of such computation can be found in [CYM03, Zak02b]

Before we leave the discuss about trees, we present the following simple theorem, which we are going to use in the next section.

Theorem 6.2.1 *Given two rooted ordered trees X and Y and the left-most subtree Y_1 of Y, let X's string representation be $x = x_1, x_2, \ldots, \#$ and that of Y_1 is $y = y_1, y_2, \ldots, y_n, \#$, if x is grater than the prefix $y' = y_1, y_2, \ldots, y_n$ of y, then $X \succ Y$.*

Proof 6.2.1 *This theorem is the direct consequence of the definition of right-most leaf and string representation of trees.*

6.3 Maximal Subgraph Mining

In the following discussion, we present a framework for mining maximal frequent subgraphs. We show that we can unify tree mining and subgraph mining into one process where we first find all frequent trees from a graph database and then construct frequent cyclic graphs from the mined trees. We developed two procedures to support this new framework. The first one is a graph partitioning method where we group all frequent subgraphs into equivalence classes based on the spanning trees they contain. The second one is a set of pruning techniques which we developed to prune graph partitions entirely or partially for efficient maximal graph mining.

6.3.1 Canonical Spanning Tree of a Graph

We define a *subtree* of an undirected graph G as an acyclic connected subgraph of G. A subtree T is a *spanning tree* of G if T contains all nodes in G. There are many spanning trees for a given graph G. We define the maximal one according to a total order defined on trees as the *canonical spanning tree* of G, denoted by $\mathcal{T}(G)$.

Example 6.3.1 *In Figure 6.3, we show an example of a labeled graph P together with all four-node subtrees of P. Each subtree is represented by its canonical representation and sorted according to the total order $\succeq$. Each such tree is a spanning tree of the graph P and the first one (T_1) is the canonical spanning tree of P.*

In the following discussion, we study the properties of canonical spanning trees. We show that the canonical spanning tree of a graph G is the maximal one among those of subgraphs of

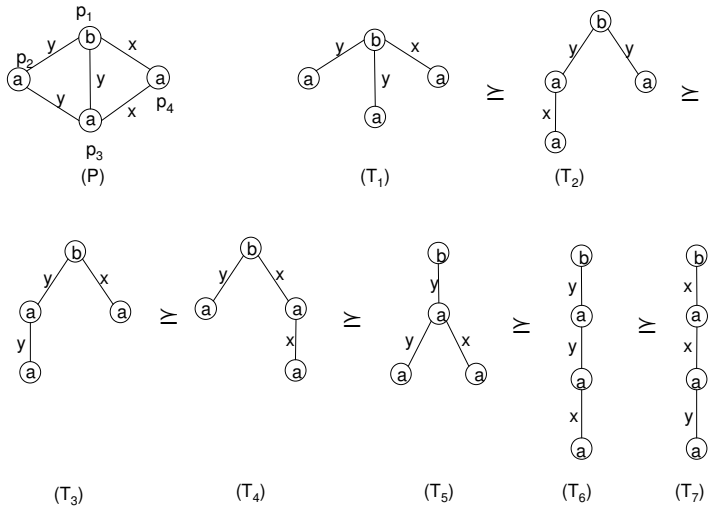

Figure 6.3: An example of a labeled graph P (upper-left), P's subtrees, spanning trees, and its canonical spanning tree (T_1).

G. Based on this property, we present a greedy algorithm to compute the canonical spanning tree of a graph. Before proving the theorem, we prove a weaker version of it on trees.

Theorem 6.3.1 *Given two free trees $R \subseteq S$, we have $\mathcal{T}(S) \succeq \mathcal{T}(R)$.*

Proof 6.3.1 *We prove the theorem by constructing a rooted, ordered version S_1 of S such that $S_1 \succeq \mathcal{T}(R)$. Upon succeeding in constructing S_1, by the definition of the canonical spanning tree, we have $\mathcal{T}(S) \succeq S_1 \succeq \mathcal{T}(R)$ and hence the proof of the theorem. To show that such S_1 always exists, let $R_1 = \mathcal{T}(R)$ be the canonical representation of R and f be a subtree isomorphism from R_1 to S. Let u be the root of R_1 and $M \subseteq V[S]$ be the set of images of nodes in $V[R_1]$ under f. We construct S_1, the rooted, ordered version of S as follows: the root of the S_1 is $f(u)$; the relative order among siblings $V \subseteq M$ follows the order of the corresponding nodes in R_1; for nodes no in M, their relative order is arbitrary but are all greater than their siblings that are in M. An example of such construction is shown in Example 6.3.2.*

To complete the proof, we show that $S_1 \succeq R_1$. To that end, let $x = x_1, x_2, \ldots, x_h$ be the string representation of R_1 and $y = y_1, y_2, \ldots, y_l$ be the string representation of S_1. We have the following observation. Let i denote the first position where $x_i \neq y_i$, we claim that we have either one of the following two conditions:

- *$x_i =$ \$ and $y_i \in \Sigma_{E[S]}$ and*
- *$x_i = \#$ and $y_i \in \Sigma_{E[S]} \cup \{\$\}$.*

In other words, at the first position i where string x and y differs, the symbol x_i can only be either a # or \$. This is because every node in R_1 has exactly one image in S_1 and hence

the first mismatch could not happen between a node or an edge label of S_1 and R_1. Given the observation and recall that every label in $\Sigma_{E[S]}$ is sorted greater than \$ and that is greater than #, we have the theorem proved.

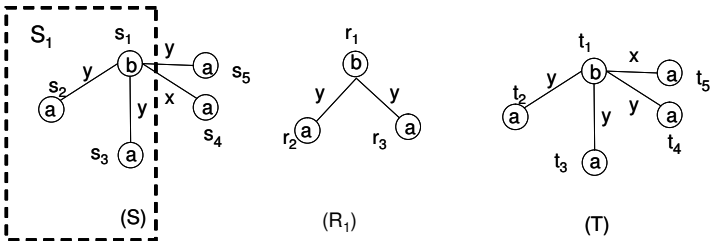

Figure 6.4: An example shows that the canonical tree of a tree S is maximal among that of all subtrees of S.

Example 6.3.2 *In Figure 6.4, we show a free tree S and an canonical representation R_1 of a free tree $R \subseteq S$. We map R_1 to a subtree S_1 of S (labeled by a dashed rectangle) using a mapping: $r_1 \rightarrow s_1$, $r_2 \rightarrow s_2$, and $r_3 \rightarrow s_3$ (there are other valid subgraph isomorphisms from R_1 to S and it is left to interested users to valid the proof if we use other subgraph isomorphisms). We construct a rooted, ordered version X of S by using s_1 as the root and s_2, s_3 as the first and the second child of s_1. The relative order of s_4 and s_5 is not important. We just choose to flip them: s_5 as the third child of s_1 and s_4 as the last child. Notice in the figure that t_4 has the edge label x to node t_1 and therefore it corresponds to the node s_5, as stated before. The string representation of R_1 is byaya\$#; that of X is byayaxaya\$# hence that we have $X \succeq R_1$. By definition we have $\mathcal{T}(S) \succeq X$ and therefore we have $\mathcal{T}(S) \succeq R_1 = \mathcal{T}(R)$.*

We prove a general form of Theorem 6.3.1 for graphs as follows.

Theorem 6.3.2 *Given two connected graphs $P \subseteq Q$, we have $\mathcal{T}(Q) \succeq \mathcal{T}(P)$*

Proof 6.3.2 *Let $R = \mathcal{T}(P)$ be the canonical spanning tree of P. We claim that we can always find a spanning tree S of Q such that $R \subset S$. The proof is straightforward. First, let $R' = R$. Second, we can find a node u in Q but not in R' and connected u to R' by an edge in Q. Such u always exists as long as R' is not the spanning tree of Q. Continue the process and finally we get a supertree of R which is a spanning tree of Q. By Theorem 6.3.1, we have $\mathcal{T}(S) \succeq R$. Therefore we have $\mathcal{T}(Q) \succeq \mathcal{T}(S) \succeq \mathcal{T}(P)$ and the proof of the theorem.*

It turns out we can prove an even stronger version of Theorem 6.3.2, which asserts that the left-most subtree S of a graph Q's spanning tree is maximal to some extents among its "peers". The theorem is stated below:

Theorem 6.3.3 *Given an n-node graph Q and a tree R that is the left-most subtree of $\mathcal{T}(Q)$, for every $(n-1)$-node subgraph $P \subset Q$, we have $R \succeq \mathcal{T}(P)$.*

Algorithm 10 Canonical Spanning Tree(G)

1: $S \leftarrow \{u|\ u$ is a single-node tree with the maximal node label in $G\}$
2: **repeat**
3: $\quad Q \leftarrow \{y|\ y$ is a subtree of G by including one additional node to a tree $x \in S\}$
4: $\quad S \leftarrow \{y|\ y \in Q, \mathcal{T}(y)$ is maximal for all tress in $S\}$
5: **until** S contain a spanning tree of G

Proof 6.3.3 *We prove the theorem by contradiction. Let's assume there is a* $(n-1)$*-node subgraph* $P \subset Q$ *such that* $S = \mathcal{T}(P) \succ R$*. We have* $\mathcal{T}(Q) \succ S$ *(by Theorem 6.3.2). Then we have* $\mathcal{T}(Q) \succ R$ *(transitivity). Finally we have* $\mathcal{T}(Q) \succ \mathcal{T}(Q)$ *(following Theorem 6.2.1). That leads to contradiction.*

Based on the theorem, we present a greedy search scheme in Algorithm 10 to calculate the canonical spanning tree of a graph G. The algorithm works by first picking up maximal labeled nodes in G as a group of single-node trees. It iteratively grows those trees by attaching an additional node to each of them in all possible ways. The resulting group of trees are inspected one by one. Only those which are maximal among their peers are selected for the next iteration. Because of Theorem 6.3.2 this scheme is guaranteed to converge to the canonical spanning tree of a graph. Since every tree is a graph, the procedure can be applied to obtain canonical representations of trees also[2].

6.3.2 Tree-based Equivalence Class

In this section based on the canonical spanning tree we introduced, we introduce a graph partitioning method.

Definition 6.3.1 ***Tree-based Equivalence Class:*** *Given two graphs* P *and* Q*, we defined a binary relation* $\cong$ *such that* $P \cong Q$ *if and only if their canonical spanning trees are isomorphic to each other i.e.* $\mathcal{T}(P) = \mathcal{T}(Q)$*. The relation* $\cong$ *is reflexive, symmetric, transitive, and hence an equivalence relation.*

Example 6.3.3 *In Figure 6.5, we show subgraphs of the graph* P *in Figure 6.1 which are not necessarily trees. Subgraphs are grouped together if they share the same canonical spanning tree. The five non-singleton groups are shown here and the remaining twelve groups are all singletons. Throughout the paper, we are only interested in subgraphs with at least an edge (i.e. excluding frequent nodes as trivial cases).*

We outline a new frequent subgraph search algorithm based on the graph partitioning method discussed hereinbefore with two steps: (1) mine all the frequent trees from a graph

[2]There is a notable difference between our procedure and the method present in [CYM03]. We do not define the center of the tree but rather find the maximal string a free tree can produce. We choose the particular way for an uniform treatment of tree and graph canonical form as stated in Theorem 6.3.3

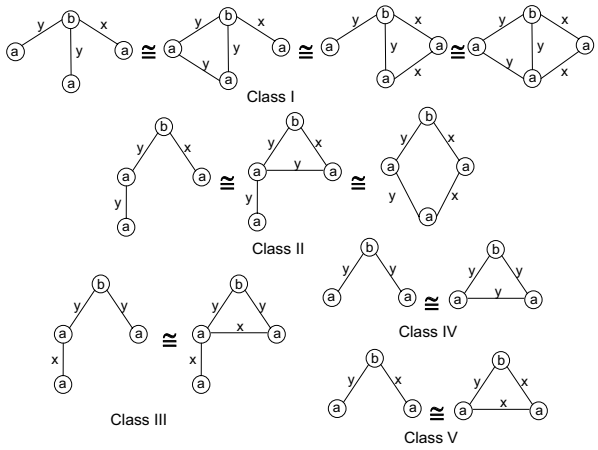

Figure 6.5: Example of tree based equivalence classes for subgraphs in graph P, presented in Figure 6.1.

database and (2) for each such frequent tree T, find all frequent subgraphs whose canonical spanning trees are isomorphic to T. Maximal frequent subgraphs can be found subsequently among frequent ones. We would like to discuss the first step later for two reasons. First, as pointed out in [YH03], the current subgraph mining algorithms can be easily tailored to find only trees from a graph database by limiting the topology of the patterns. This is true for Closegraph [YH03] as well as for FFSM [HWP03], which is our previously developed depth-first subgraph mining algorithm. Second, most of the techniques developed for mining subtrees from a forest can also be easily adapted for the same purpose. Therefore, in the following discussion, we focus on step 2, which is how to enumerate the equivalence class of a tree T. The two-step division of the mining algorithm is artificially introduced to make it easy to explain the key ideas of the algorithm without exceeding details.

6.3.3 Enumerating Graphs from Trees

We define a *joining* operation $\oplus$ between a graph (tree) G and a hypothetical edge connecting any two nodes i, j in G with label e_l such that $G \oplus (i, j, e_l) = G'$ where G' is a supergraph of G with one additional edge between nodes i and j with label e_l. If the graph G already contains an edge between nodes i and j, the joining operation fails and produces nothing. If G' is frequent, we denote the hypothetical edge (i, j, e_l) as a *candidate edge* for G. The above definition can serve as the basis for a recursive definition of the joining operation between a graph G and a candidate edge set $E = \{e_1, e_2, \ldots, e_n\}$ such that $G \oplus E = (G \oplus e_1) \oplus \{e_2, \ldots, e_n\}$.

Let's assume we already calculated the set of candidate edges $C = \{c_1, c_2, \ldots, c_n\}$ from the set of all possible frequent hypothetical edges. We define the *search space* of G, denoted by $G : C$, as the set of graphs which might be produced by joining the graph G and a candidate

Algorithm 11 Maximal Subgraph Mining($\mathcal{G}$, σ)

1: $\mathcal{R} \leftarrow \{T | T$ is a frequent tree in $\mathcal{G}\}$
2: $S \leftarrow \{G | G \in$ Expansion(T), $T \in \mathcal{R}\}$
3: return $\{G | G \in S$, G is maximal $\}$

edge set in the powerset set of C (denoted by 2^C). That is:

$$G : C = \{G \oplus y | y \in 2^C\} \tag{6.1}$$

In the following discussion, the group of candidate edges are sometimes referred to as the "tail" of the graph G in its search space. We present a recursive algorithm in Algorithm 13 to enumerate the search space for a graph G and the overall frequent subgraph search algorithm is presented in Algorithm 11.

Example 6.3.4 *in Figure 6.6, we single out the largest equivalence class (Class One) from Figure 6.5. We show a tree K with its tail $C = \{(k_2, k_3, y), (k_3, k_4, x)\}$. K's search space is composed of four graphs $\{K, K_{S1}, K_{S2}, K_{S3}\}$ (K is always included in its search space) and is organized into a "search tree" in analogy to frequent item set mining. This tree structure follows the recursive procedure we present in Procedure 12 which can be used to explore the search space for a given graph.*

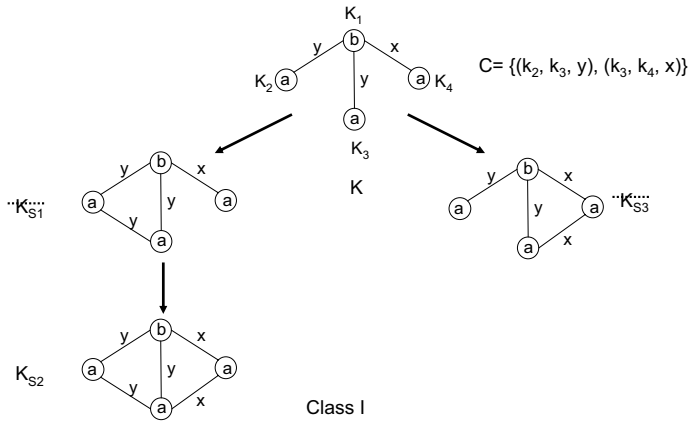

Figure 6.6: Example of enumerating graph's search space. We use dashed lines on the subgraph K_{S1} and K_{S3} to denote the fact that they will be pruned by an optimization technique which is discussed in Section 6.3.6

6.3.4 Enumerating Frequent Trees from a Graph Database

We outline a generic depth-first tree enumerating method in Algorithm 14. The algorithm begins by finding all frequent edges from a graph database. The recursive function Generic-

Procedure 12 Expansion(T)

1: $C \leftarrow \{c|\ c$ is a candidate edge for $T\}$
2: $S \leftarrow$ Search Graphs (T, C)
3: **return** $\{G|\ G \in S$, G is frequent, G has the same canonical spanning tree as T has$\}$

Procedure 13 Search Graphs($G, C = \{c_1, c_2, \ldots, c_n\}$)

1: $Q \leftarrow \emptyset$
2: **for each** $c_i \in C$ **do**
3: $\quad Q \leftarrow Q \cup$ Search Graphs$(G \oplus c_i,\ \{c_{i+1}, c_{i+2} \ldots, c_n\})$
4: **end for**
5: **return** Q

Algorithm 14 Generic-Tree-Explorer($C,\ R$)

1: $Q \leftarrow \emptyset$
2: **for each** each $X \in C$ **do**
3: $\quad S \leftarrow \{Y|Y$ is a frequent tree of one additional node of $X \in C\}$
4: $\quad S \leftarrow S - R$ # avoiding duplicated search
5: $\quad (U, V) \leftarrow$ Generic-Tree-Explore(S, R)
6: $\quad Q \leftarrow Q \cup U \cup$ Expansion(X) #including cyclic graphs
7: $\quad R \leftarrow R \cup \{X\} \cup V$
8:
9: **end for**
10:
11: **return** (Q, R)

Tree-Explorer is invoked to perform a depth-first search where set C holds all trees that are to be enumerated (candidates) and set R holds all trees that have been enumerated. For a given n-node candidate X, all its frequent $n+1$-node supertrees Y are included as the candidates at the next level. Duplicated candidates are detected and removed at line 7. By adding the method to enumerate the search space of a tree (line 9) the algorithm is in fact enumerating all frequent graphs. The updated sets are returned where set Q holds the group of frequent graphs and R holds the set of frequent trees. Finally, maximal frequent subgraphs are computed from set of frequent subgraphs.

6.3.5 Putting it together

Our method to enumerate cyclic graphs is independent of the tree mining method and hence that people can freely choose any tree mining method in the implementation. The one we used, which is considerably more complicated and more efficient, is based on the FFSM algorithm [HWP03] and is detailed in Appendix.

In summary, we developed a different method to enumerate frequent subgraphs. The

Algorithm 15 Maximal Subgraph Mining($\mathcal{G}$, σ)

1: $C \leftarrow \{c|\ c$ is a frequent edge in $\mathcal{G}\ \}$
2: $(\mathcal{F}, \mathcal{S})$= Generic-Tree-Explorer($C$, $\emptyset$)
3: **return** $\{G|G \in \mathcal{F}$ and G is maximal $\}$

method may be slightly more efficient than previous ones[3] but the technique for maximal subgraphs involves inefficient postprocessing. In the next section, we introduce optimization techniques to improve maximal frequent subgraphs search.

6.3.6 Optimizations: Globally and Locally Maximal Subgraphs

In this section by developing optimization techniques, we demonstrate that maximal subgraphs can be mined efficiently. Our developed techniques dynamically remove a set of frequent subgraphs that can not be maximal from a search space. To that end, we define a frequent subgraph G to be *locally maximal* if it is maximal in its equivalence class i.e. G has no frequent supergraph(s) that share the same canonical spanning tree as that of G; we refer to a subgraph as *globally maximal* if it is maximal frequent in a graph database. Clearly, every globally maximal subgraph must be locally maximal but not vice versa. Our pruning techniques aim to avoid enumerating subgraphs which are not locally maximal.

Bottom-Up Pruning

The search space of a graph G is exponential in the cardinality of the candidate edges set C. One heuristic to avoid such an exponential search space is to check whether the largest possible candidate $G' = G \oplus C$ is frequent or not. If G' is frequent, each graph in the search space is a subgraph of G' and hence not maximal. This heuristics is referred to as the *Bottom-Up Pruning* and can be applied to every step in the recursive search procedure presented in Algorithm 12. By applying the bottom-up pruning to the equivalence class I presented in Figure 6.5, graph K_{S1} and K_{S3} are pruned.

Dynamic Reordering. An important technique related to the efficiency of the bottom-up pruning is the *dynamic reordering* technique, which works in two ways. First, it trims infrequent candidate edges from the tail of a graph to reduce the size of the search space (an edge candidate can become infrequent after several iterations since other edges are incorporated into the patterns). Second, it rearranges the order of the elements in the tail according to their support value. For example, given a graph's tail C, by dynamic reordering, we sort the elements in C by their support values, from lowest to highest. After this sorting, the infrequent "head" is trimmed. At the end of the remaining tail is a family of elements individually

[3]tree related operations are treated separately from graph related operations and therefore save processing time

having high support and hence the pattern obtained by grouping them together is likely to still have high support value. This heuristics is widely used in mining maximal itemsets to gain performance. However, without the spanning tree framework, applying dynamic ordering is very difficult in any of the current subgraph mining algorithms, which intrinsically have a fixed order of adding edges to an existing pattern for various performance considerations.

Tail Shrink

Given a graph G and a supergraph G' of G, an *embedding* of G in G' is a subgraph isomorphism f from G to G'. We prefer the term embedding to subgraph isomorphism, though interchangeable, for more intuitive descriptions. In Figure 6.7, we show a subgraph L and its supergraph P. There are two embeddings of L in P: $(l_1 \rightarrow p_1, l_2 \rightarrow p_2, l_3 \rightarrow p_3, l_4 \rightarrow p_4)$ and $(l_1 \rightarrow p_1, l_2 \rightarrow p_3, l_3 \rightarrow p_2, l_4 \rightarrow p_4)$. We define a candidate edge (i, j, e_l) to be *associative* to a graph G if it appears in every embedding of G in a graph database. In other words, a candidate edge (i, j, e_l) of G is associative if and only if for every embedding f of G in a graph G', G' has the edge $(f(i), f(j))$ with label e_l. One example of an associative edge is the edge (l_1, l_3, y) to the tree L shown in Figure 6.7.

If a tree T contains a set of associative edges $\{e_1, e_2, \ldots, e_n\}$, any maximal frequent graph G which is a supergraph of T must contain all such edges. Hence we can remove these edges from the tail of T and augment them to T with no risk of missing any maximal ones. This technique is referred to as the *tail shrink* technique. Tail shrink has two advantages: (1) it reduces the search space and (2) it can be used to prune the entire equivalence class in certain cases. To elaborate the latter point, we define a set of associative edges C of a tree T to be *lethal* if the resulting graph $G' = T \oplus C$ has a canonical spanning tree other than that of T i.e. $\mathcal{T}(G') \neq \mathcal{T}(T)$.

We formally prove the correctness of the tail-shrinking optimization by the following theorem. First we prove that if a tree T contains a group A of associative candidates, any maximal supergraph of T must contains all such candidates. If the group of associative candidates is lethal, we prove that any maximal supergraph G of T can not be in T's equivalence class.

Theorem 6.3.4 *Given a tree T, its tail C, a group of associative edge $A \subseteq C$ of T, and a maximal graph $G \supset T$, we have $G \supseteq T \oplus A$ (I). Further more, if A is lethal, we have $\mathcal{T}(G) \succ \mathcal{T}(T)$ (II).*

Proof 6.3.4 *We prove (I) by showing contradiction. Let's assume there is a G such that $G \supset T$ and $G \not\supseteq T \oplus A$, then there is at least one $c = (i, j, e_l) \in A$ that is not part of G. Since c occurs in every instance of T, it occurs in every instance of G. Then we can construct a $G' = G \oplus c \supset G$ which has the same support value of G and hence frequent. This contradicts to the fact that G is maximal. For (II), we have $\mathcal{T}(G) \succeq \mathcal{T}(T \oplus A)$ (by Theorem 6.3.2) and $\mathcal{T}(T \oplus A) \succ \mathcal{T}(T)$ (by the definition of lethal). Therefore we have $\mathcal{T}(G) \succ \mathcal{T}(T)$.*

Example 6.3.5 *In Figure 6.7, we show an example of an associative edge $e = (1, 3, y)$ of L which is lethal to L also. In the same example, the lethal edge e can be augmented to each member of the class II to produce a supergraph with the same support and with a canonical spanning tree sorted greater than L. Therefore the whole class can be pruned away once we detect a lethal edge(s) to the tree L. Interested reader might verify that the equivalence class III shown in Figure 6.5 also has a lethal edge and the whole class can be pruned away by tail shrink.*

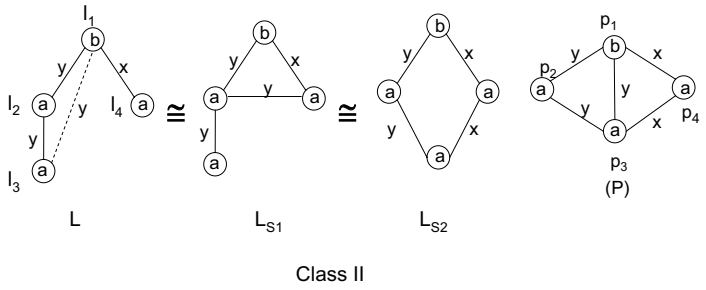

Figure 6.7: An example that shows how tail shrink might be used to prune the whole equivalence class. Edge $e = (l_1, l_3, y)$, denoted by a dashed line to be distinguished from other edges, is associative to tree L and lethal to L as well. The graph obtained by joining L and e should belong to equivalence class I shown in Figure 6.5.

External-Edge Pruning

In this section, we introduce a technique to remove one equivalence class without any knowledge about its candidate edges. We refer to this technique as the *external-edge pruning*. We define an edge to be an *external edge* for a graph G if it connects a node in G and a node which is not in G. We represent an external edge as a three-element tuple (i, e_l, v_l) to stand for the fact that we introduce an edge with label e_l incident on the node i in a graph G and a node which does not belong to G with node label v_l. An external edge (i, e_l, v_l) is *associative* to a graph G if and only if:

- for every embedding f of G in a graph G', G' has a node v with the label v_l and is not an image of a node in $V[G]$ under f, and
- v connects to the node $f(i)$ with an edge label e_l in G'

.

Example 6.3.6 *We show two examples of associative external edges in Figure 6.8. One is (m_1, x, a) for the tree M and another one is $(n_f 1, y, a)$ for the tree N. If a tree T has at least one associative external edge, the entire equivalence class of T can be pruned since the same*

edge can be augmented to every member of the class. In this example, both equivalence classes IV and V can be eliminated due to the external-edge pruning.

Once we find that a tree T has an associative external edge, the equivalence class associated with tree T can be eliminated since the same edge can be augmented to each members in T's equivalence class and therefore none of them are maximal.

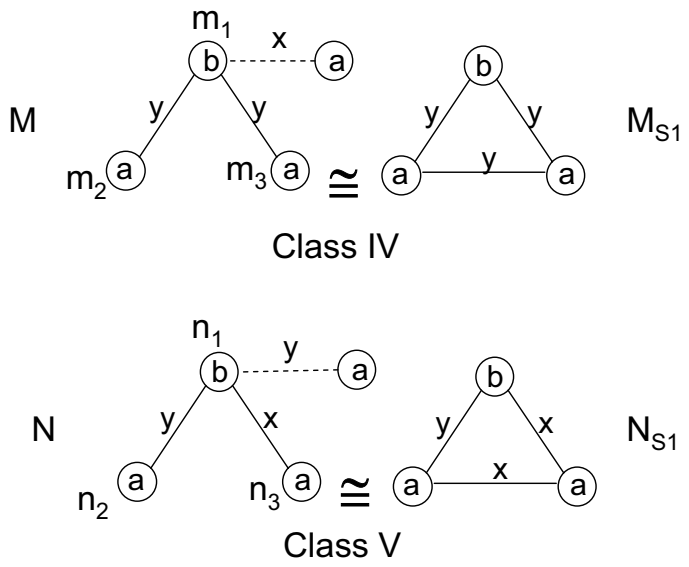

Figure 6.8: Examples showing external edges and associative external edges.

In summary, we present three pruning techniques to speed up maximal subgraph mining. For the graph P shown in Figure 6.1, there are a total of twenty five subgraphs of P, including itself and excluding the null graph. These subgraphs are partitioned into five non-singleton classes, shown in Figure 6.5, and twelve singleton classes (not shown). There is only one maximal subgraph, namely, graph P itself. We have successfully pruned every one of the five non-singleton equivalence classes (P of the equivalence class I is left untouched since it is maximal). What we do not show further is that we can apply the same techniques to the remaining twelve singleton equivalence classes to eliminate all of them. Interested readers might verify that themselves.

Algorithm 16 and Algorithm 19 integrate these optimizations into the basic enumerate technique we presented in Algorithm 11 and Algorithm 12.

6.3.7 Computational Details

In this section, we discuss computational details that are important for an efficient implementation of the SPIN algorithm. First, we present two algorithms to support the optimizations we designed for SPIN, namely, computing the edge candidates for a tree and determining

Algorithm 16 MaxSubgraph-Expansion(T) An algorithm for exploring the equivalence class of a tree T for maximal subgraph mining

1: $C \leftarrow \{c|\ c$ is a candidate edge for $G\}$
2: $A \leftarrow \{c|\ c \in C$ and c is associative $\}$
3: **if** A is lethal **then** return$\emptyset$ **end** # *tail shrinking*
4: $S \leftarrow$ Search Graphs ($T \oplus A$, $C - A$)
5: return $\{G|\ G \in S$, G is frequent and maximal, $\mathcal{T}(G) = \mathcal{T}(T)\}$

Algorithm 17 Search-graphs($G, C = \{c_1, c_2, \ldots, c_n\}$)

1: **if** $G \oplus C$ is frequent **then** return $\{G \oplus C\}$ **end** # *bottom-up pruning*
2: $Q \leftarrow \emptyset$
3: **for each** $c_i \in C$ **do**
4: $\quad Q \leftarrow Q \cup$ Search-graphs($G \oplus c_i$, $\{c_{i+1}, c_{i+2} \ldots, c_n\}$)
5: **end for**
6: **return** Q

Algorithm 18 Maximal Subgraph Mining($\mathcal{G}$)

1: $C \leftarrow \{c|\ c$ is a frequent edge in $\mathcal{G}\}$
2: $(\mathcal{M}, S)=$ Generic-Tree-Explorer(C, $\emptyset$)
3: **return** $\mathcal{M}$

Algorithm 19 Generic-Tree-Explorer(C, R)

1: $Q \leftarrow \emptyset$
2: **for each** $X \in C$ **do**
3: $\quad S \leftarrow \{Y|Y$ is a frequent tree of one additional node of $X \in C\}$
4: $\quad S \leftarrow S - R$ # *avoiding duplicated search*
5: $\quad (U, V) \leftarrow$ Generic-Tree-Explore(S, R)
6: $\quad Q \leftarrow Q \cup U, R \leftarrow R \cup \{X\} \cup V$
$\quad$ # *external-edge pruning*
7: $\quad$ **if** X has no external associative edge **then**
8: $\quad\quad Q \leftarrow Q \cup$ Max-subgraph-Expansion(X) # *mining cyclic graphs*
9: $\quad$ **end if**
10: **end for**
11: **return** (Q, R)

whether there is an external associated edge for a frequent tree pattern. We then present two procedures to calculate a pattern's support value and make sure it is maximal. By those procedures, we guarantee that every pattern reported by SPIN is frequent and maximal.

How to Calculate the Candidate Edge Set?

To calculate the candidate edge set for a tree T, we need to scan all embeddings of T and count how many graphs having an edge candidate e for every possible such edge. If T has n nodes, there are up to $n * (n-1)/2 * |\Sigma_E|$ edges we need to monitor where $|\Sigma_E|$ is the total

number of edge labels in a graph database. This computation is used frequently in the overall search and we want to optimize it as much as possible.

The optimization relies on the depth-first tree search method we introduced. Assuming an edge $e = (i, j, e_l)$ that connects two nodes in a tree T' that is the parent node of the tree T in a depth-first tree enumeration algorithm, one necessary condition for e to be frequent in T is that it must be frequent in T'. In other words, an edge e must be a candidate edge in T' in order to be considered as a candidate edge in T. Thus, to speed up the candidate edge calculation, T can "inherit" all candidate edges from its parent, verify their frequencies, and select those that are frequent. The additional edges T needs to be considered are those that are not part of T', namely, edges that connect the newly introduced node of T and nodes in T'.

How to Find an External Associative Edge?

Determining whether there is an associative external edge for a tree T can be very expensive. This involves scanning every embedding of T and searching for the associative edges. With depth-first search we can use an efficient heuristic. Given a tree pattern T, assuming tree T' is T's parent node in a depth-first enumeration, a *hint* of T is an associative external edge e of T'. When we check the tree T, we first check its hints (assuming there is at least one from its parent node). The reason is that if an edge is associative to a tree T', it is very likely to be associative to its child node also. If at least a hint e is valid, we prune the whole equivalence class of T. During such search process, we might come to situations where we "run out of hints", i.e. there is no valid associative external edge from its parent node for a tree T. In such cases, we employ an expensive search procedure to search every embedding of T to determine the set of associative external edges.

How to Calculate the Support Value for a Cyclic Graph?

Given a correct tree enumeration algorithm, the only thing we need to make sure about the SPIN algorithm is that every cyclic graphs is indeed frequent. A brute force way to guarantee this is performing a linear scan of the graph database and using subgraph isomorphism test to determine the support of a cyclic graph. It turns out we have a much efficient way to perform such calculation than the brute force one. Given a graph database $\mathcal{G}$, a tree T, and an edge candidate e in T's tail C, the *embedding set* e_s of e is the group of embeddings of graph $T \oplus e$ in $\mathcal{G}$. Given a cyclic graph $G = T \oplus E = \{e_1, e_2, \ldots, e_n\}$, the embedding set of G can be computed efficiently by intersecting the embedding sets of cyclic graphs $T \oplus e$ for all $e \in E$. The support of G is the fraction of graphs those embeddings map to. This is further explained in the following example.

Example 6.3.7 *Graph K has two embeddings in the graph database $\mathcal{G} = \{P, Q, R\}$ showing*

in Figure 6.1. Those are $\{(k_1 \rightarrow p_1, k_2 \rightarrow p_2, k_3 \rightarrow p_3, k_4 \rightarrow p_4), (k_1 \rightarrow p_1, k_2 \rightarrow p_3, k_3 \rightarrow p_2, k_4 \rightarrow p_4)\}$. *Given* $e_1 = (k_2, k_3, y)$ *and* $e_2 = (k_3, k_4, x)$ *the cyclic graph* $K \oplus e$ *has the same two embeddings while the cyclic graph* $K \oplus e_2$ *has only the first embedding of* K. *Therefore graph* $K \oplus \{e_1, e_2\})$ *has only one embedding:*$(k_1 \rightarrow p_1, k_2 \rightarrow p_2, k_3 \rightarrow p_3, k_4 \rightarrow p_4)$.

Where does the efficiency come from? First, we notice that we do not need to perform any subgraph isomorphism checking. Second, we do not even need to scan the database since if we keep the embedding set of each candidate edge, the only operation we need to do is set intersection. Given candidate edge set $E = \{e_1, e_2, \ldots, e_n\}$, computing the support value of a cyclic graph $G = T \oplus E$ is bounded by $O(|E| \times n)$ where n is the total number of embeddings T has. In our implementation, we index embeddings of a tree T by integers 1, 2, ..., n. The embedding set of any cyclic graph $T \oplus e$ can be efficiently encoded as a bit-vector $V = v_1, v_2, \ldots, v_n$ where v_i is the 1 if and only if $T \oplus e$ has the ith embedding of T. The set operation can be efficiently performed by the bitwise and operation.

How to Find Maximal Subgraphs?

We notice that with the optimizations we presented, all reported graphs G are locally maximal, i.e. there is no supergraph of G sharing the same canonical spanning tree. To select globally maximal subgraphs, we need to check whether there is an extern edge e such that if attached, the resulting graph (or tree) is frequent. If there exists such external edge, the graph is not maximal. Otherwise, it is. This is similar to what we did in searching for external associated edge and we could use similar strategy to save computation.

6.4 Experimental Study

We performed our empirical study using a single processor of a 2.8GHz Pentium Xeon with 512KB L2 cache and 2GB main memory, running RedHat Linux 7.3. The SPIN algorithm is implemented using the C++ programming language and compiled using g++ with O3 optimization. We compared SPIN with two alternative subgraph mining algorithms: FFSM ([HWP03]) and gSpan [YH02]. Every maximal subgraph reported by SPIN in synthetical and real data sets are cross validated using results from FFSM and gSpan to make sure it is (a) frequent, (b) maximal, and (c) unique.

6.4.1 Synthetic Dataset

To evaluate the performance of the SPIN algorithm, we first generate a set of synthetic graph databases using a synthetic data generator [KK01]. There are six parameters to control the behavior of the synthetic graph generator and we summarize them in Table 6.1. The synthetic data generator works by first creating a set of candidate graphs (the total number

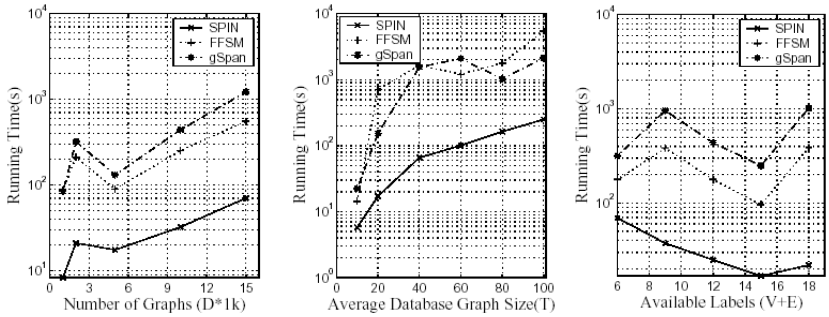

Figure 6.9: Left: performance comparison with different database sizes (D) using SPIN, FFSM and gSpan. Middle: the same comparison using different average database graph sizes (T) and Right: using different number of available labels. We fix the support value to be 1% in all cases. When we change the total number of labels, we fix the ratio between node labels and edge labels at 1:2

are controlled by L) with user specified size (I). The candidates are subsequently selected, replicated and augmented with random selected node and edges to create a graph database. Further details of the synthetic graph generator might be found in [KK01].

Parameter	meaning
D	total number of graphs in a generated database
T	average graph size (edges)
L	the total number of potentially frequent subgraphs
I	the size of the potentially frequent subgraphs (edges)
V	the total number of node labels
E	the total number of edge labels

Table 6.1: We list the set of parameters controlling the behavior of the synthetic graph generator. We use a single string, e.g. "$D10kT30L200I11V4E4$" to represent a graph database generated by a particular combination of parameters. For example, "$D10kT30L200I11V4E4$" stands for a synthetic graph database which contains a total of $D = 10$k (10000) graphs. On average, each graph contains $T = 30$ edges with up to $V = 4$ vertex labels and $E = 4$ edge labels. There are total $L = 200$ potential frequent patterns in the database with average edge number $I = 11$. We use a fixed value $L = 200$ throughout the paper following previous published papers.

In Figure 6.10, we represent the performance comparison of SPIN, FFSM, and gSpan algorithms for a synthetic data set with different support values. When the support is set to a pretty high value e.g. 5%, the performance of all three algorithms are pretty close. SPIN scales much better than the other two algorithms as we decrease the support values. At support value 1%, SPIN provides a six and ten fold speed-up over FFSM and gSpan, respectively. We do not show data with support value great than 5% since there is little difference among the three methods.

We then launched a set of four experiments to test the scalability of SPIN for important parameters listed in Table 6.1. In Figure 6.11, we compare the performance using different value of I, which specifies the average size of the set of subgraphs to be embedded frequently in a synthetic graph database. These subgraphs are referred to as *seeds* in [KK01] and they

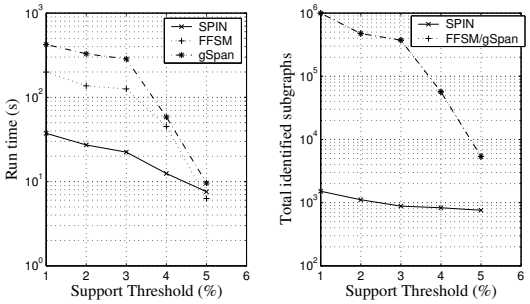

Figure 6.10: Left: performance comparison under different support values for data set $D10kT30L200I11V4E4$ using SPIN, FFSM and gSpan. Here we follow the common convention of encoding the parameters of a synthetic graph database as a string. Right: Total frequent patterns identified by the algorithms.

correlate to the set of maximal frequent subgraphs we need to find. The results demonstrate that SPIN scales well with different sizes of seeds. On the other hand, FFSM and gSpan have to enumerate all subgraphs of these seeds. They clearly experience an exponential growth of the total number of such subgraphs (shown in the right part of Figure 6.11) as the sizes of the created seeds increase. We notice more than an order of magnitude speed-up in the study and the performance gap can become even larger as we further increase the value of I.

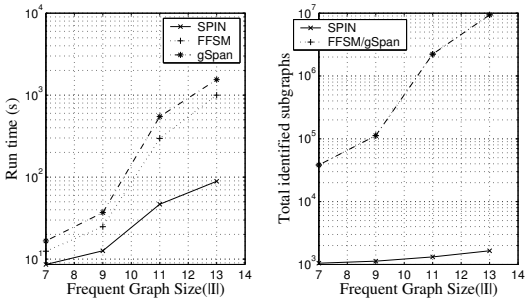

Figure 6.11: Left: performance comparison with different seed sizes(I) using SPIN, FFSM and gSpan. Right: Total frequent patterns identified by the algorithms.

In Figure 6.9, we further compared the scalability of the SPIN using different database sizes, different average graph transaction sizes, and different number of node/edge labels. Throughout the experimental studies, SPIN always offers a huge performance gain comparing to FFSM and gSpan.

6.4.2 Chemical Data Set

We also applied SPIN to two widely used chemical data sets to test its performance. The data sets are obtained from the DTP AIDS Antiviral Screen test, conducted by U.S. National Cancer Institute. In the DTP data set, chemicals are classified into three sets: confirmed active (**CA**), confirmed moderately active (**CM**) and confirmed inactive (**CI**) according

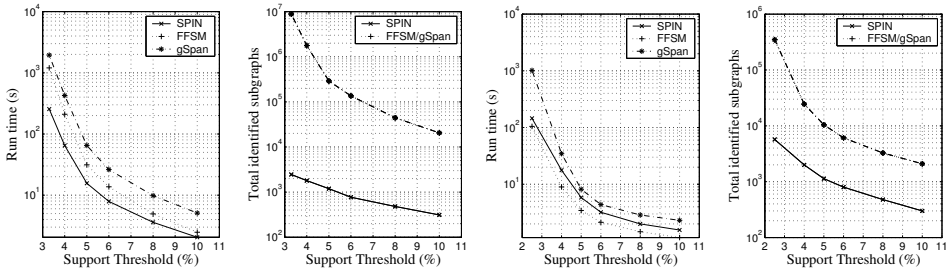

Figure 6.12: Left: performance comparison under different support values for DTP CA data set using SPIN, FFSM and gSpan. Right: Total frequent patterns identified by the algorithms.

to experimentally determined activities against the HIV virus. There are in total of 423, 1083, and 42115 chemicals in the three sets, respectively. For our own purposes, we used all compounds from CA and from CM to form two data sets, which are subsequently referred to as DTP CA and DTP CM, respectively. The DTP data can be downloaded from http://dtp.nci.nih.gov/docs/aids/aids_data.html. The characteristics of these two data sets are summarized in Table 6.2.

In Figure 6.12, we show the performance comparison of SPIN, FFSM, and gSpan using the DTP CA data set. We report that SPIN is able to expedite the program up to five(eight) fold, comparing with FFSM(gSpan) at support value 3.3%. Mining only maximal subgraphs can reduce the total number of mined patterns by a factor up to three orders of magnitude in this data set. We also applied the same algorithms to the data set DTP CM. In this case, SPIN has a performance very close to FFSM and both are around eight fold speed-up over gSpan. However, if we impose an additional constraint to let FFSM output the maximal patterns it finds among the set of frequent patterns, SPIN offers a three fold speed-up from FFSM.

Data Set	N	$\overline{V}$	$\overline{L_V}$	$maxV$	$minV$	$\overline{E}$	$\overline{L_E}$	$maxE$	$minE$
DTP CA	423	39.56	3.99	190	10	42.2	2.058	196	10
DTP CM	1083	31.8	3.665	220	6	34.25	2.07	234	4

Table 6.2: The characteristics of the two data sets. N is the total number of graphs in a data set; $\overline{V}$ and $\overline{E}$ specify the average numbers of vertices and edges; $\overline{L_V}$ and $\overline{L_E}$ specify the average numbers of vertex labels and edge labels; and $maxV/minV$ and $maxE/minE$ are the maximal/minimal numbers of vertices/edges of all graphs in a data set, respectively.

6.5 Conclusions

In this chapter we present SPIN, an algorithm to mine maximal frequent subgraphs from a graph database. A new framework, which partitions frequent subgraphs into equivalence classes is proposed together with a group of optimization techniques. Compared to current state-of-the-art subgraph mining algorithms such as FFSM and gSpan, SPIN offers very good

scalability to large graph databases and at least an order of magnitude performance improvement in synthetic graph data sets. The efficiency of the algorithm is also confirmed by a benchmark chemical data set. The algorithm of compressing large number of frequent subgraphs to a much smaller set of maximal subgraphs will help us to investigate demanding applications such as finding structure patterns from proteins in the future.

Chapter 7

Constraint-Based Subgraph Mining

7.1 Introduction

In discovering frequent patterns in graph represented task-relevant data, one important component is being able to incorporate a constraint in the mining process. Typically we will meet the following constraints [HK01]:

- Data constraints: these specify the property of the data. For example, most of the chemical structure graphs are spare (close to trees). This property may be used to derive efficient algorithms.
- Dimension constraints: these specify the complexity of the pattern (e.g. the size of the subgraph).
- Interestingness constraints: these specify the measure that used to quantify the interestingness of a graph pattern.

The first two types of constraints have been discussed in previous chapters. In this chapter, we study the interestingness constraint. The task of formulating the right constraint(s) is left for domain experts. As part of our computational concern, we answer the following two questions: (1) what types of constraints can be efficiently incorporated into a subgraph mining algorithm and (2) how to incorporate a constraint if it can be efficiently incorporated. The answers to the two questions are the major contribution of this chapter and is discussed in detail in this chapter.

This chapter is organized as follows. In Section 7.2, we review definitions related to graphs and introduce the constrained subgraph mining problem. In Section 7.3, we present a detailed description of our method. For simplicity, we concentrate on cliques in all the section and the justification for this is presented in Section 7.4 when we describe mining multilabeled graphs. Since the major application of the constraint based subgraph mining is to discover structure patterns in protein structures, the experimental study of the constraint-based subgraph mining is presented in Section 9.

7.2 Constraint-based Frequent Subgraph Mining

7.2.1 Examples of Constraints

Here a *constraint* is a function that maps subgraphs to boolean values. By convention the value true implies that a subgraph has some desired properties in a domain and otherwise not. For example, the following statement "in protein structure motif discovery, each amino acid residue in a structure motif must have a solvent accessible surface of sufficient size" is a constraint. This constraint selects only those structure motifs that are close to the surface of proteins. We use symbol ρ to denote a constraint.

7.2.2 The Constrained Subgraph Mining Problem

Given a constraint ρ and a support threshold σ, the problem of *constrained subgraph mining* is to identify all frequent subgraph C in a graph database $\mathcal{G}$ such that $\rho(C)$ is true. For simplicity, we only study cliques in the following example. Figure 7.1 shows all cliques (without any constraint) that appear in at least two graphs in the graph database shown in the same figure. If we use support threshold $\sigma = 2/3$ without any constraint, all six cliques are reported to a user. If we increase σ to 3/3, only cliques A_1, A_2, A_3, and A_4 are reported. If we use support threshold $\sigma = 2/3$ and the constraint that each clique should contain at least one node with the label "a", the constrained frequent cliques are A_1, A_3, A_4, and A_6.

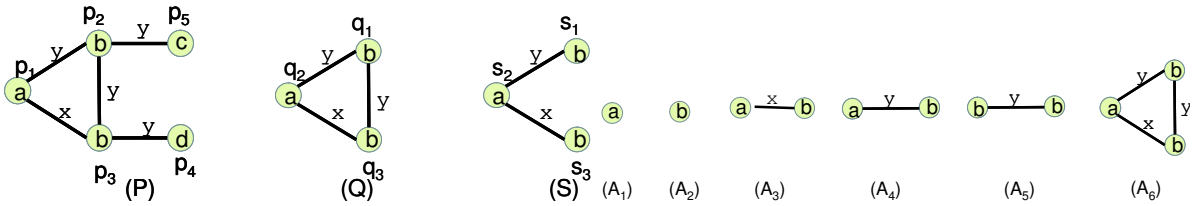

Figure 7.1: Left: a database $\mathcal{G}$ of three labeled graphs. Right: all (non-empty) frequent cliques with support $\geq 2/3$ in $\mathcal{G}$. The actual support values are: (3/3, 3/3, 3/3, 3/3, 2/3, 2/3) for cliques from A_1 to A_6.

7.3 The Constrained Subgraph Mining Algorithm

In this section, we present a detailed discussion on (1) what types of constraints can be incorporated efficiently into a subgraph mining algorithm and (2) how to incorporate them.

Our strategy relies on designing graph normalization functions that map subgraphs to strings. As discussed in Chapter 5, a desirable property of a string graph normalization function is prefix-preservation. To briefly review the terminology, we repeat the some of the definitions before.

- a string graph normalization function is a 1-1 mapping from graphs to strings.

- For a graph G, its image with a string graph normalization is called the *canonical code* of G.

- A string graph normalization is *prefix-preserving* if for all graph G, there exists a subgraph $G' \subset G$ such that the canonical code of G' is a prefix of the canonical code of G.

Examples of prefix-preserving graph normalization functions include the DFS code [YH02] and the CAM code [HWP03]. As we prove in Theorem 7.3.8, with a generic depth first search algorithm, a prefix-preserving graph normalization function guarantees that no frequent constrained patterns can be missed. The design challenge here is to construct a graph normalization function that is prefix-preserving in the presence of constraints.

In the following discussion for simplicity, we only concentrate on cliques (fully connected subgraphs). Section 7.4 provides a justification of concentrating on cliques. As proved in Theorem 7.4.1, using multiple labeled graphs, algorithms that discover cliques can be used to discover general subgraphs as well.

7.3.1 A Synthetic Example of Constraints

We use the following constraint as our driving example for constrained clique mining. The constraint states that we should only report frequent cliques that contain at least one edge label of "y". The symbol "y" is selected to make the constraint works best with the graph examples that we show in Figure 7.1. With the threshold σ=2/3 and applying this constraint to all the frequent cliques shown in Figure 7.1, we find that only three cliques satisfy the constraint, namely A_4, A_5, and A_6.

In the following we call the constraint *edge label constraint* and show a graph normalization function that is prefix-preserving for the edge label constraint. Before we do that, we introduce a normalization that does not support any constraints. Our final solution adapts this constraint-unaware graph normalization function.

7.3.2 A Graph Normalization Function That Does Not Support Constraints

We use our previous canonical code [HWP03] for graph normalization, outlined below for completeness.

Given an $n \times n$ adjacency matrix M of a graph G with n nodes, we define the *code* of M, denoted by *code*(M), as the sequence of lower triangular entries of M (including the node labels as diagonal entries) in the order: $M_{1,1}M_{2,1}M_{2,2}...M_{n,1}M_{n,2}...M_{n,n-1}M_{n,n}$ where $M_{i,j}$ represents the entry at the ith row and jth column in M. Since edges are undirected, we are concerned only with the lower triangular entries of M. Figure 7.2 shows examples of adjacency matrices and codes for the labeled graph Q shown in the same figure.

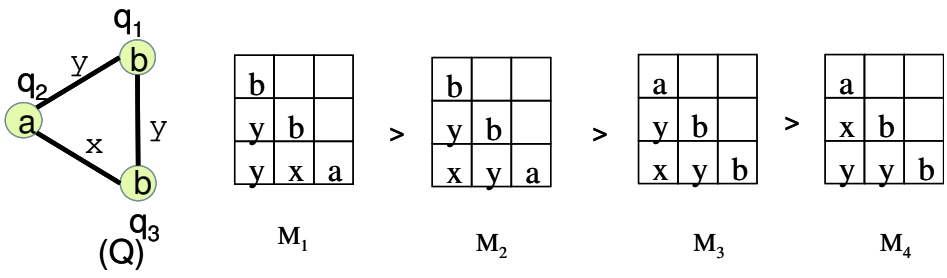

Figure 7.2: All possible adjacency matrices for the clique Q. Since adjacency matrices for undirected graph are symmetric, only the lower triangular part of the matrices are shown. Using lexicographic order of symbols: $y > x > d > c > b > a > 0$, we have $code(M_1) =$"$bybyxa$" $> code(M_2) =$ "$bybxya$" $> code(M_3) =$ "$aybxyb$" $> code(M_4) =$"$aybxyb$". Hence $code(M_1)$ is the canonical code for the graph Q.

The lexicographic order of sequences defines a total order over adjacency matrix codes. Given a graph G, its *canonical code*, denoted by $\varphi(G)$, is the maximal code among all its possible codes. For example, $\varphi(M_1) =$"$bybyxa$" shown in Figure 7.2 is the canonical code of the graph.

7.3.3 A Graph Normalization Function that Supports Constraints

In this section, we introduce a graph normalization ψ that we call "generic". In Section 7.3.4, we show that several widely used constraints for protein structure motifs can be well handled by ψ. In Section 7.3.5, we show how to use the function ψ in a depth-first constrained clique search procedure to make sure that each constrained frequent clique is enumerated exactly once.

Definition 7.3.1 *Given the graph normalization function $\varphi$with its codomain Γ and a constraint ρ, a* ***generic graph normalization function*** *ψ, is a function that maps a graph G to Γ^* recursively as:*

$$\psi(G) = \begin{cases} \varphi(G) & \text{if } \rho(G) \text{ is false} \\ \varphi(G) & \text{if } |V(G)| = 1 \\ \max\limits_{G' \subset G, \rho(G')} \psi(G')\$\varphi(G) & \text{otherwise} \end{cases}$$

where G' is a subgraph of G with size one less than G and $\psi(G')\$f(G)$ is the concatenation of the code produced by $\psi(G')$, the symbol \$, and the code $F(G)$. We assume that the symbol \$ is not in the set Γ and we use this symbol to separate different parts of the generic graph normalization. The total ordering on strings (the max function) is defined by the string lexicographical order with the underlying symbol ordering assumed from Γ.

Example 7.3.1 *With the edge label constraint, the graph normalization φ, and the generic graph normalization, for the graph Q shown in Figure 7.2, we show that $\psi(Q) = b\$byb\$bybyxa$.*

First, bybyxa is a suffix of the code since it is the canonical code of graph Q. Second, in a search for $\psi(Q)$, two subgraphs of Q that satisfy the edge label constraint are considered. One is a single edge, with an edge label "y", that connects two nodes both with the label "b". The other is also a single edge, with an edge label "y", that connects nodes with labels "a" and "b". Since the canonical code for the first ("byb") is greater than that of the second ("bya"), we choose the first subgraph and concatenate it canonical code "byb" before the string "bybyxa". We obtain a partial result "byb$bybyxa". Finally we add a single symbol "b" before the partial result "byb$bybyxa" at the last step of the recursive definition. Therefore we have $\psi(G) =$ bbybbybyxa.

Theorem 7.3.1 *$\psi(G)$ exists for every graph G with the edge label constraint.*

Proof 7.3.1 *Let's assume that a graph G satisfy the edge constraint, i.e. G contains an edge label "y". For if the graph G does not contain any edge label "y", its code is $\varphi(G)$, which is well defined.*

We claim that for all G that contains at least one edge label "y", there exists a subgraph G' of G that also contains the same label. This observation suggests that we can find at least one G' in the recursive definition. Applying the reasoning recursively, we conclude that $\psi(G)$ is well defined.

Theorem 7.3.2 *ψ is a 1-1 mapping and thus a graph normalization function.*

Proof 7.3.2 *If two graphs are isomorphic, they must give the same string, according to the definition of ψ. To prove that two graphs that have the same canonical strings are isomorphic, we notice that the last element of the code produced by ψ is $\varphi(G)$. Since $\varphi$is a graph normalization procedure, two identical sequence must imply the same graph.*

Theorem 7.3.3 *For all G such that $\rho(G)$ is true, there exists a subgraph $G' \subset G$ with size one less than G such that $\rho(G')$ is true and $\psi(G')$ is a prefix of $\psi(G)$.*

Proof 7.3.3 *This property is a direct result of the recursive definition 7.3.1.*

We notice that in proving Theorems 7.3.2 and 7.3.3, we do not use the definition of the constraint ρ. In other words, Theorems 7.3.2 and 7.3.3 can be proved as long as we have Theorem 7.3.1. Therefore, we have the following Theorem:

Theorem 7.3.4 *If ψ is defined for every graph with respect to a given constraint ρ, ψ is 1-1 and prefixing-preserving.*

Proof 7.3.4 *This is a direct result of the recursive definition 7.3.1.*

7.3.4 More Examples Related to Protein Structure Motifs

To show the utility of the generic graph normalization ψ, we examine several widely used constraints and show that ψ can handel them well.

Our first example is a constraint that is widely used in structure motif discovery. The *connected component constraint* (CC constraint for short) asserts that in a structure motif, each amino acid residue is connected to at least another amino acid residue by a contact relation and that the motif is a connected component with respect to the contact relation. The intuition of the CC constraint is that a structure motif should be compact and hence has no isolated amino acid residue. To be formal, the CC constraint is a function *cc* that assigns value true to a graph if it is a connected component according to the contact relation and false otherwise.

Our second example is the *contact density constraint*, which asserts that the ratio of the number of contacts and the total number of edges in a structure motif should be greater than a predefined threshold. Such ratio is referred to as the *contact density* (density) of the motif and the constraint is referred to as the density constraint. The intuition of the density constraint is that a structure motif should be compact and the amino acid residues in the motif should well interact with each other. This constraint may be viewed as a more strict version of the CC constraint which only requires a motif to be a connected component. Again, to be formal, the density constraint is a function d that assigns value true to a graph if its contact density is at least some predefined threshold and otherwise false.

It would be an awkward situation if we need to define a new graph normalization for each of the constraints we discuss above. Fortunately, this is not the case. In the following discussion, we show that generic graph normalization function ψ is well defined for these two constraints.

Theorem 7.3.5 *$\psi(G)$ exist for every graph G with respect to the CC constraint or the density constraint.*

Proof 7.3.5 *We only show the proof for the CC constraint and that for the density constraint can be proved similarly. The key observation is that for every connected component G of size n, there exists a subgraph $G' \subseteq G$ with size $n-1$ such that G' is also a connected component. The observation is a well-known result from graph theorem and a detail proof can be found in [HWP03].*

Following Theorem 7.3.4, we have the following corollary.

Corollary 7.3.6 *ψ is a 1-1 mapping and prefix-preserving for the CC constraint or the density constraint.*

Proof 7.3.6 *This is obtained directly from Theorem 7.3.4.*

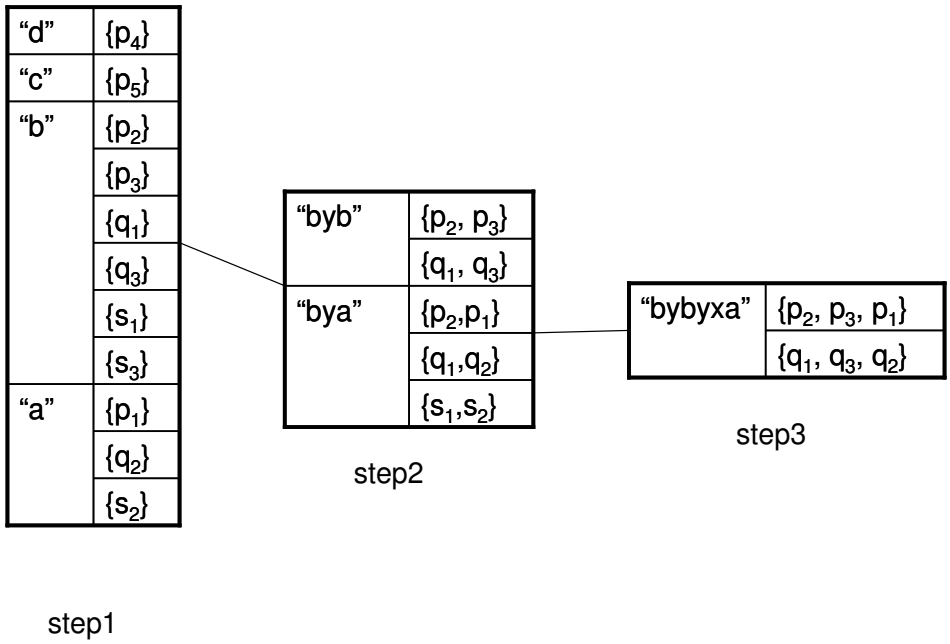

Figure 7.3: The contents of the hash table counter after applying the CliqueHashing algorithm to the data set shown in Figure 7.1 with the edge label constraint.

After working with several example constraints, we study the sufficient and necessary condition such that the graph normalization function ψ is well defined for an arbitrary constraint ρ. The following theorem formalizes the answer.

Theorem 7.3.7 *Given a constraint ρ, $\psi(G)$ exist for all graphs G with respect to the constraint ρ if and only if for each graph G of size n such that $\rho(G)$ is true, there exists a subgraph $G' \subset G$ of size $n-1$ such that $\rho(G')$ is also true.*

Proof 7.3.7 *(if) For all graphs G such that $\rho(G)$ is true, if there exists one $G' \subset G$ such that $\rho(G')$ is also true, by the definition of ψ, $\psi(G)$ exists.*

(only if) If $\psi(G)$ exists for every graph G with respect to a constraint ρ, for a graph G such that $\rho(G)$ is true, by the definition of ψ, we always have at least one $G' \subset G$ such that $\rho(G')$ is also true.

7.3.5 CliqueHashing

We have designed an efficient algorithm, Algorithm 20, that identifies all frequent cliques from a labeled graph database with constraints, as described below.

At the beginning of the algorithm we scan a graph database and find all frequent node types (line 1-4 in Algorithm 20). The node types and their occurrences are kept in a hash table *counter*. At a subsequent step, a frequent clique with size $n \geq 1$ is picked from the hash

table and is extended to all possible $n+1$ cliques by attaching one node to the occurrences in all possible ways (line 1-3 Algorithm 21. The same Algorithm is used below). These newly discovered cliques and their occurrences are again indexed in a separate hash table (line 4-10) and enumerated recursively (line 11-15). The algorithm backtracks to the parents of a clique if no further extension from the clique is possible (line 16). The overall algorithm stops when all frequent node types have been enumerated. In the algorithm, we the following notations. Symbol $\mathcal{F}$denotes all frequent constrained patterns. $x \sqsubseteq y$ if string x is a prefix of string y. $s(G)$ is the support of a graph G.

Figure 7.3 illustrates the CliqueHashing algorithm with the edge label constraint.

Theorem 7.3.8 *If ψ is well defined for all possible graphs with a constraint ρ, the CliqueHashing algorithm enumerates all frequent constrained cliques from a graph database exactly once.*

Proof 7.3.8 *The prefix preserving property of Definition 7.3.1 implies that at least one subclique of a frequent clique will pass the IF statement of line 12 in the backtrack_search procedure in CliqueHashing. Therefore the algorithm will not miss any frequent cliques in the presence of a constraint ρ.*

The proof that the algorithm discovers every constrained frequent cliques exactly once may not be obvious at first glance. The key observation is that for a clique G of size n, there is only one subclique with size $n-1$ that has a code matching a prefix of $\psi(G)$. If we can prove the observation, by the line 12 of the backtrack_search procedure, the CliqueHashing algorithm guarantees that each constrained frequent cliques will be discovered exactly once.

To prove the observation, we assume to the contrary that there are at least two such subcliques with the same size and both give codes as prefixes of $\psi(G)$. We claim that one of the two codes must be a prefix of the other (by the definition of prefix). The claim leads to the conclusion that one of two subcliques must be a subclique of the other (by the definition of ψ). The conclusion contradicts our assumption that the two subcliques have the same size.

7.4 CliqueHashing on Multi-labeled Graphs

A *multi-labeled graph* is a graph where there are two or more labels associated with a single edge in the graph. The CliqueHashing algorithm can be applied to multi-labeled graphs directly without major modifications. The key observation is that our enumeration is based on occurrences of cliques (line 3 in function backtrack_search). In Figure 7.4, we show a graph database with three multi-labeled graphs. In same figure, we show (pictorially) how the CliqueHashing algorithm can be applied to graphs with multilabels.

In the context of the structure motifs detection, handling multi-labeled graphs is important for the following reason. First, due to the imprecision in 3D coordinates data in motif

Algorithm 20 CliqueHashing($\mathcal{G}$, σ, ρ)

1: **for each** node label $t \in \lambda(v), v \in V[G], G \in \mathcal{G}$ **do**
2: $counter[t] \leftarrow counter[t] \cup \{v\}$
3: $C \leftarrow C \cup \{t\}$
4: **end for**
5: **for** $t \in C$ **do**
6: **if** $s(t) \geq \sigma$, $\rho(s)$ is true **then**
7: $\mathcal{F} \leftarrow \mathcal{F} \cup$ backtrack_search$(t, counter[t])$
8: **end if**
9: **end for**
10: **return** $\mathcal{F}$

Algorithm 21 backtrack_search($t_0, \mathcal{O}$, ρ)

1: **for each** clique $h \in \mathcal{O}$ **do**
2: $\mathcal{O}' \leftarrow \{f | f = h \cup v, h \subset V[G], v \in (V[G] - h)\}$
3: **end for**
4: $C \leftarrow \emptyset$
5: $counter \leftarrow \emptyset$
6: **for each** occurrence of a clique $f \in \mathcal{O}'$ **do**
7: $t \leftarrow \psi(f)$
8: $counter[t] \leftarrow counter[t] \cup \{f\}$
9: $C \leftarrow C \cup \{t\}$
10: **end for**
11: **for each** $t \in C$ **do**
12: **if** $s(t \geq \sigma$, $t \sqsubseteq t$, and $\rho(t)$ is true **then**
13: $\mathcal{F} \leftarrow \mathcal{F} \cup$ backtrack_search$(t, counter[t])$
14: **end if**
15: **end for**
16: **return** $\mathcal{F}$

discovery, we need to tolerate distance variations between different instances of the same motif. Second, partitioning the 1D distance space into distance bins is not a perfect solution since distance variations can not be well handled at the boundary of the bins. In our application distance bins may lead to a significant number of missing motifs. Using a multi-labeled graph we can solve the boundary problem by using "overlapping" bins to take care of boundary effect.

The following discussion provides the connection between cliques and general subgraphs.

7.4.1 Link Cliques to General Subgraphs

In the following discussion, we present the connection between algorithms that identify all cliques and those that identify subgraphs with arbitrary topology. To that end, we introduce two two graph transformations to transform the graph search space:

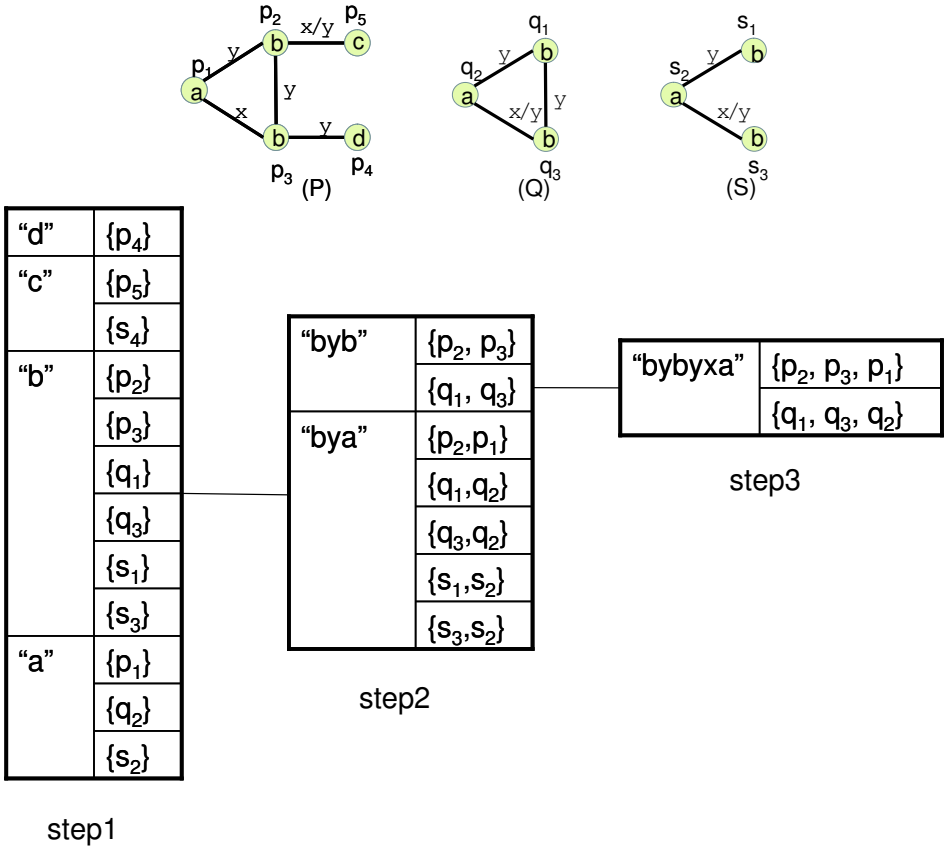

Figure 7.4: Top: a graph database of three graphs with multiple labels. Bottom: the contents of the hash table counter after applying the CliqueHashing algorithm to the data set shown left.

Definition 7.4.1 *Given a graph space $G*$ with a symbol set Σ and a symbol $\$ \notin \Sigma$, we define a graph transformation Π: $G^* \rightarrow G'^*$ defined on $\Sigma' = \Sigma \cup \{\$\}$ such that $\Pi(G) = G'$ where:*

- $G = \{V, E, \lambda, \Sigma\}$,
- $G' = \{V', E', \lambda', \Sigma'\}$,
- $V' = V$,
- $E' = \{(u, v) \mid u, v \in V, u \neq v\}$,
- $\Sigma' = \Sigma \cup \{\$\}$,
- $\lambda'(u) = \lambda(u)$ *for* $u \in V$,
- $\lambda'(e) = \begin{cases} \lambda(e) & \textit{for all } e \in E \\ \$ & \textit{otherwise} \end{cases}$

Similarly, we define another transformation Π': $G^* \rightarrow G'^*$ such that $\Pi'(G) = G'$ where:

- $G = \{V, E, \lambda, \Sigma\}$,
- $G' = \{V', E', \lambda', \Sigma'\}$,
- $V' = V$,
- $E' = \{(u, v) \mid u, v \in V, u \neq v\}$,
- $\Sigma' = \Sigma \cup \{\$\}$,
- $\lambda'(u) = \lambda(u)$ *for* $u \in V$,
- $\lambda'(e) = \lambda(e) \cup \{\$\}$ *for* $e \in E$

The composition transformation $\Pi' \circ \Pi(G)$ is defined as $\Pi' \circ \Pi(G) = \Pi'(\Pi(G))$ and is abbreviated by $\Pi'\Pi$. Without causing confusion, the same set operators: Π, Π', and, $\Pi'\Pi$ are defined on graph databases by transforming each of its element such that: $\Pi(\mathcal{G}) = \{\Pi(G) \mid G \in \mathcal{G}\}$, $\Pi'(\mathcal{G}) = \{\Pi'(G) \mid G \in \mathcal{G}\}$, and $\Pi'\Pi(\mathcal{G}) = \{\Pi'\Pi(G) \mid G \in \mathcal{G}\}$.

Theorem 7.4.1 *Given a graph database* $\mathcal{G}$*, we have:*

- G *is a frequent induced subgraphs of* $\mathcal{G}$ *if and only if* $\Pi(G)$ *is a frequent clique in* $\Pi(\mathcal{G})$,
- G *is a frequent subgraphs of* $\mathcal{G}$ *if and only if* $\Pi(G)$ *is a frequent clique in* $\Pi'\Pi(\mathcal{G})$.

Proof 7.4.1 *This is a direct results of the subgraph matching definition we presented before.*

In the previous a few chapters, we present a few subgraph mining algorithms that identify subgraph patterns in a collection of graphs. In the next a few chapters, we show how the techniques can be used to obtain patterns in protein structures.

Chapter 8

Identifying Structure Pattern in Proteins

8.1 Introduction

This chapter presents results for the following protein structure pattern discovery problem: given a set $\mathcal{G}$ of three dimensional (3D) protein structures, identify all structure patterns that occur with sufficient frequency among the proteins in $\mathcal{G}$.

Though the motivations of the pattern discovery problem have been discussed in previous chapters such as Chapter 1, we want to echo the same points and summarize them below:

- Known 3D protein structures grows rapidly. Currently there are $> 35,000$ known 3D protein structures in the Protein Data Bank (PDB, [BWF$^+$00]) and the number grows exponentially in the past several years. Further more, the recent Structural Genomics projects [SFK00] aim to generate many new protein structures in a high-throughput fashion, which is expected to accelerate the already high rate of protein structures determination. With fast growing structure data, automatic and effective knowledge discovery tools are needed in order to generate testable hypotheses about the functional role of proteins and the evolutionary relationship among proteins. Identifying structure pattern represents a useful way towards such goals [GAW97, MSO03, SKK02, WKHK04].

- There is a complex relationship between protein structure and protein function [EJT04]. It is well known that global structure similarity does not necessarily imply similar function. For example, the TIM barrels are a large group of proteins with remarkably similar global structures, yet widely varying functions [NOT02]. Conversely, similar function does not necessarily imply similar global structure: the most versatile enzymes, hydrolyases and the O-glycosyl glucosidases, are associated with 7 different global structural families [HG99]. Because of the puzzling relationship between global protein structure and function, recent research effort in protein structure comparison has shifted to identifying local structural patterns that are responsible for biological functions. These

functions include (but not limited to) protein-protein interaction, protein-ligand binding, and catalysis [BKB02, BT03, SR03, SSR04, TJ04, TCCV03].

For recent reviews of methods and applications of protein structure pattern discovery, see [JT04].

In this chapter using a graph representation of proteins, we have formalized the structure pattern identification problem as a frequent clique mining problem in a set of graphs $\mathcal{G}$ with certain additional constraints. Some constraints are encoded in the graph representation of protein structure as pair-wise amino acid residue distances, pair-wise amino acid residue interactions, and the physico-chemical properties of the amino acid residues and their interactions in a protein structure. Other constraints enforce that structure patterns should be a connected components according to the pairwise interaction relationship and are discussed in detail in Section 8.6.

Compared to other methods, our method offers the following advantages.

- First, our method is efficient. It usually takes only a few seconds to process a group of proteins of moderate size (e.g. 30 proteins), which makes it suitable for processing protein families defined by various classifications such as SCOP or EC (Enzyme Commission).

- Second, our results are specific. As we show in our experimental study section, by requiring structure pattern to occur frequently among a group of proteins rather than common to two proteins, we significantly reduce spurious patterns without losing patterns with clear biological relevance. With a quantitative definition of significance based on the hyper-geometric distribution, we find that the structure patterns that we identify are specifically associated with the original family. This association may significantly improve the accuracy of feature-based functional annotation of structures from structural genomics projects.

The rest of the chapter is organized in the following way. Section 8.2 presents our way of mapping protein structures to labeled graphs. Section 8.3 describes a variety of distance based methods to determine the contact relationship between a pair of amino acid residues, which is an important problem in mapping protein structures to graphs. Section 8.4 study the efficiency of different graph representations. Case studies of several protein families are presented in Section 8.6. The statistical significance of the structure patterns are studied in Section 8.5.

8.2 Representing Protein Structure by Labeled Graph

We model a protein structure as a labeled undirected graph in the following way. The nodes represent amino acid residues in a protein structure. We create edges connecting each and

every pair of (distinct) residues, labeled by two types of information: (1) The Euclidian distance between the two C_α atoms in the residues and (2) A boolean indicates the physico-chemical contact relationship between two residues. We compute the distance between two C_α atoms since the protein backbone defines the overall protein conformation.

Precisely, a protein in our study is a labeled graph $P = (V, E, \Sigma, \lambda)$ where

- V is a set of nodes that represents the set of amino acid residues in the protein
- $E = V \times V$ - (u, u) for all $u \in V$
- $\Sigma = \Sigma_V \cup \Sigma_E$ is the set of disjoint node labels (Σ_V) and edge labels (Σ_E)
- Σ_V is the set of 20 amino acid types
- $\Sigma_E = \mathbb{R}^+ \times \{true, false\}$ where $\mathbb{R}^+$ is the set of positive real numbers
- $\lambda : V \cup E \rightarrow \Sigma_V \cup \Sigma_E$ assigns labels to nodes and edges.

In practice we are not concerned with structure patterns with size over certain long distances (say > 13 Å), so proteins need not be represented by complete graphs. Since each amino acid occupies a real volume, the number of edges per node in the graph representation can be bounded by a small constant. Our graph representation can be viewed as a hybrid of two popular representations of proteins: that of distance matrix [EJT04] and that of protein contact map [HSS+02].

Our graph representation is similar to those used by other groups [Rus98, WTR+03]. The major difference is that in our representation, geometric constraints are in the form of pairwise distances and are embedded into the graph representations. The absence of geometric constraints can lead to many spurious matches as noticed in [Rus98, WTR+03]. From our experience pairwise distances handles geometric constraints well.

Comparing two popular representations of protein structures: point set representations used by geometric hashing techniques and labeled graph representations used here, we notice that in labeled graph representations, we may explicitly specify the pairwise contact relationship. This is an advantage of labeled graphs since pairwise contacts offer additional information. Taking advantage of the information, we incorporate various constraints in pattern discovery to further reduce irrelevant patterns.

In the following, we discus two important topics in creating labeled graph representation for proteins: (1) how to discretize distances into distance bins, and (2) how to define pairwise residue contacts.

8.2.1 Distance Discretization

To map continuous distances to discrete values, we discretize distances into bins. The width of such bins is commonly referred to as the *distance tolerance*, and popular choices are

$l \leq 4$	$4 < l \leq 5.5$	$5.5 < l \leq 7$	$7 < l \leq 8.5$
1	2	3	4
$8.5 < l \leq 10$	$10 < l \leq 11.5$	$11.5 < l$	
5	6	7	

Figure 8.1: Mapping distances l to bins. The unit is Å.

1 Å [MSO03], 1.5 Å [BKB02], and 2 Å [SKK02]. In our system, we choose the median number 1.5 Å as shown in Figure 8.1, which empirically delivers patterns with good geometric conservation.

8.3 Defining Contacts of Amino Acid Residues

We compute contact relationship between two residues using the following three approaches.

8.3.1 Distance Based Contact Definition

Our first definition is that two amino acid residues are in contact if their distance is no more than a certain threshold δ. The distance between two residues is defined as the Euclidian distance between the C_α atoms of the two residues. Since in principle we are interested in defining nearest neighbor residues located within the physical interaction distance, we chose δ from 6.5 Å to 8.5 Å as the threshold with the default value 8.5 Å. The value is chosen such that above the range people believe that physical or chemical interactions between amino acids occur unlikely. We refer to contacts obtained by this distance-threshold dependent method as the CD contacts.

8.3.2 Delaunay Tessellation

We have applied a technique from computational geometry to define pairwise residue contacts. In our study, two residues are in *contact* if they can be connected by a Delaunay edge [STV96]. This tessellation [Del34] is defined for a finite set of points (each point is the C_α atoms of a residue) by an *empty sphere property*: A pair of points is joined by an edge if and only if one can find an empty sphere whose boundary contains those two points. The Delaunay captures neighbor relationships in the sense that there is a point in space that has the chosen two points as closest neighbors. The Delaunay is dual to the Voronoi diagram—two points are joined by an edge in the Delaunay if and only if their Voronoi cells share a common face. Figure 8.2 illustrates the Delaunay in 2D with solid lines, and the dual Voronoi with dashed). We used the Quickhull [BDH96] program to compute the Delaunay edges. We further confine the

contact edges to ones with associated distances up to a up-limit ranging from 6.5 Å to 8.5 Å. We refer to contacts obtained in this way as the DT contacts.

8.3.3 Almost Delaunay Tessellation

The definition of the Delaunay tessellation depends on the precise coordinate values given to its points, but we know that these coordinate values are not exact in the case of proteins due to measurement imprecision and atomic motions. Thus, Bandyopadhyay and Snoeyink recently defined the almost-Delaunay edges [BHL$^+$05] by relaxing the empty sphere property to say that a pair of points p and q is joined by an almost-Delaunay edge with parameter ϵ, or $AD(\epsilon)$, if by perturbing all points by at most ϵ, p and q can be made to lie on an empty sphere. Equivalently, they look for a shell of width 2ϵ, formed by concentric spheres, so that p and q are on the outer sphere, and all points are outside the inner sphere. All Delaunay edges are in $AD(0)$, and $AD(\epsilon) \subseteq AD(\epsilon')$ for $\epsilon \leq \epsilon'$. Therefore, the almost-Delaunay edges are a superset of the Delaunay edges, whose size is controlled by the parameter ϵ. Various values of the parameter ϵ correspond to different allowed perturbations or motions. 0.1–0.25 Å would model decimal inaccuracies in the PDB coordinates or small vibrations, and 0.5–0.75 Å would model perturbations due to coarser motions. Thus, the protein graphs constructed with the almost-Delaunay edges are termed AD graphs. The precise parameter value for each edge of the contact distance graph can be computed by an algorithm that is much like the roundness algorithms from the computer-aided design (CAD) field of computational metrology. Code is available from http://www.cs.unc.edu/~debug/papers/AlmDel, or see [BHL$^+$05] for algorithmic details.

8.4 Comparing Graph Representations

8.4.1 Experimental setup

To exclude redundant structures from our analysis, we used the culled PDB list (http://www.fccc.edu/research/labs/dunbrack/pisces/culledpdb.html) with sequence similarity cutoff

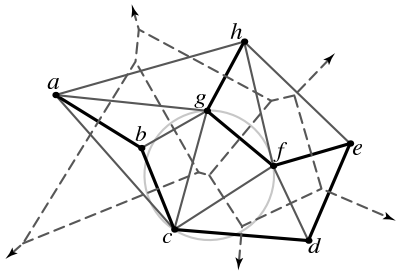

Figure 8.2: Examples of a Voronoi diagram and its dual Delaunay Tessellation for 2D points [a–f]

value 90% (resolution = 3.0, R factor = 1.0). This list contains about one quarter of all protein structures in PDB; remaining ones in PDB are regarded as duplicates to proteins in the list. We study four SCOP families: Eukaryotic Serine Protease (ESP), Papain-like Cysteine Protease (PCP), Nuclear Binding Domains (NB), and FAD/NAD-linked reductase (FAD). Each protein structure in a SCOP family was converted to its graph representation as outlined in Section 8.2. The pairwise amino acid residue contacts are obtained by computing the almost-Delaunay edges [BS04]. Structure patterns from a SCOP family were identified using the CliqueHashing algorithm with the CC constraint that states "each amino acid residue in a motif should contact at least another residue and the motif should be a connected component with respect to the contact relation". All computations were run on a single processor, 2.8GHz Pentium PC with 2GB memory, operating on RedHat Linux 7.3. The frequent subgraph mining algorithm was implemented in C++ and compiled using g++ with O3 optimization.

8.4.2 Graph Density

Three contact maps, CD, DT and AD, were constructed for each protein. Figure 8.3 shows the average number of edges per vertex as a function of the distance threshold for the Kinase dataset. For small distance thresholds, the graphs are nearly the same. As the distance threshold grows, the number of edges in CD graphs grows as a third power of the distance, while it remains almost constant in DT graphs. The number of $AD(\epsilon)$ edges as a function of the distance interpolates between the CD and DT representations.

We obtained the coordinates for all proteins used in our studies from the Protein Data Bank (PDB). Three graph representations: CD, DT and AD were constructed for each protein using methods stated above. Figure 8.3 shows the average number of edges per vertex as a function of the threshold distance. For small distances the graphs are nearly the same, but as the distance grows, the number of edges in CD grows cubically, while DT remain almost constant and $AD(\epsilon)$ interpolates between CD and DT.

8.4.3 Performance Comparison

Using a group of serine proteases, we have compared the performance of the subgraph mining algorithm using four different graph representations.

- **CD**: distance-based protein contact maps with distance threshold 10 Å.
- **AD**: almost-Delaunay based protein contact maps with distance threshold 10 Å and perturbation threshold 0.1 Å.
- **GC**: the same as CD but all pairwise amino acid residue distances are computed.

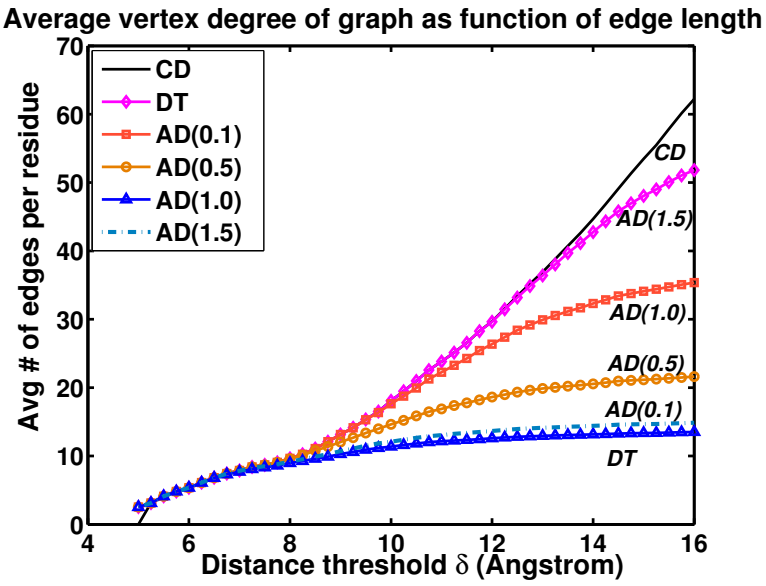

Figure 8.3: Average vertex degree as a function of the distance threshold for the three types of graphs representing the cytoplasmic domain of the Type I Tgf- Receptor in complex with Fkbp12 (1bc6).

- **AD+GC**: the same as AD but but all pairwise amino acid residue distances are computed.

Figure 8.4 shows the CPU time and the total number of patterns identified as a function of the support threshold.

As expected, CD graphs have the slowest running time. Reducing the number of contacts, AD graphs improve the performance of CD graphs with around an order of magnitude speedup. Encoding geometric constraints, GC graphs represent an alternative way to improve CD graphs with more than an order of magnitude speedup. Combining AD graphs and GC graphs together, the AD+GC graphs improve further the running time. In all the experiments, the known pattern of the serine protease family (the catalytic triad for serine proteases) are present in the results from all of the graph representations. Hence we do not loose important patterns in improving computational efficiency.

8.5 Statistical Significance of Motifs

We evaluated empirically the statistical significance of structure patterns we obtain. To that end, we devised two method. The first method randomly samples proteins from the protein structure space and apply our pattern mining algorithm to search for patterns. The experiments are repeated many times to estimate the probability that we observe at least one pattern using randomly selected proteins. The lower the probability is, the higher confidence we have about the searched structural patterns.

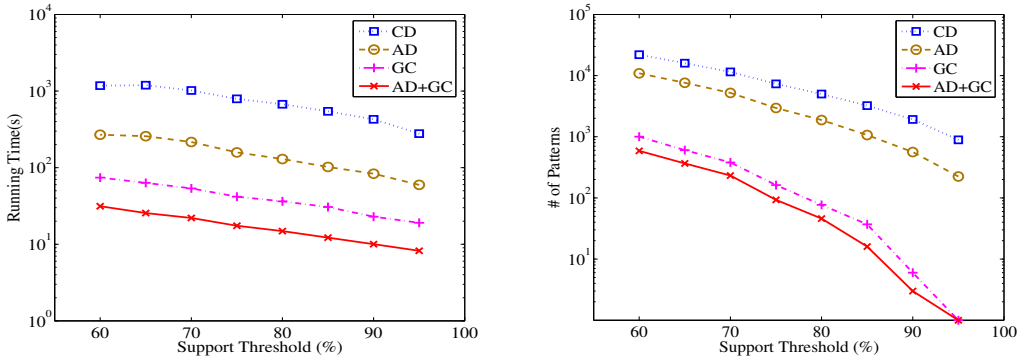

Figure 8.4: The total running time of the subgraph mining algorithm (left) and the total number of identified patterns. The support threshold varies from 60% to 90%, with 5% increase each time.

The second method computes the statistical significance of each structure pattern, using a hyper-geometric distribution. The detail of the two methods are below.

8.5.1 Estimating Significance by Random Sampling

In our experimental study, we randomly sampled 20 proteins (without replacement) from an none-redundant pdb list [WD03] and applied our algorithm to search for patterns with support ≥ 15. The parameters 20 and 15 were set up to mimic a typical size of a SCOP family. We repeated the experiment 100,000 times, and did not find a single recurring geometric pattern. Limited by the available computational resources, we did not test the system further; however, we are convinced that the chance of observing a random spatial motif in our system is rather small.

8.5.2 Estimating Significance by Hyper-Geometric Distribution

We estimate the statistical significance of a structural motif m by computing the P-value associated with its occurrences in an existing protein family. To that end, we obtain a group of representative structures M, computed as the Culled PDB list [WD03], as a sample of the protein structure population (all possible protein structures, crystalized or not).

Our null hypothesis H_0 is that the pattern m randomly occurs in the protein structure population. Given an existing protein family $F \subset M$, a set of proteins $S \subseteq M$ where m occurs, the probability of observing a set of at least k proteins in F contain m under the null hypothesis is given by the following hyper-geometric distribution [BKB02]:

$$P\text{–}value = 1 - \sum_{i=0}^{k-1} \frac{\binom{|F|}{i}\binom{|M|-|F|}{|T|-i}}{\binom{|M|}{|T|}}. \tag{8.1}$$

where $|X|$ is the cardinality of a set X. For example, if a pattern m occurs in every member

pattern	Composition	κ	δ	$-log(P)$	pattern	Composition	κ	δ	$-log(P)$
Eukaryotic Serine Protease (ID: 50514) N: 56 σ: 48/56, T: 31.5									
1	DHAC	54	13	100	20	DHAC	50	6	100
2	ACGG	52	9	100	21	HACA	50	8	100
3	DHSC	52	10	100	22	ACGA	50	11	100
4	DHSA	52	10	100	23	DSAG	50	16	100
5	DSAC	52	12	100	24	SGGC	50	17	100
6	DGGG	52	23	100	25	AGAG	50	27	95
7	DHSAC	51	9	100	26	AGGG	50	58	85
8	SAGC	51	11	100	27	ACGAG	49	4	100
9	DACG	51	14	100	28	SCGA	49	6	100
10	HSAC	51	14	100	29	DACS	49	7	100
11	DHAA	51	18	100	30	DGGS	49	8	100
12	DAAC	51	32	99	31	SACG	49	10	98
13	DHAAC	50	5	100	32	DSGC	49	15	98
14	DASC	49	20	92	33	SAGG	49	31	90
15	DGGL	49	53	83	34	DSAGC	48	9	99
16	DSSC	48	12	97	35	SCSG	48	19	93
17	AGAG	48	19	93	36	SAGG	48	23	88
18	DSGS	48	23	94	37	DAAG	48	27	89
19	DASG	48	32	87	38	GGGG	48	71	76
Papain-like cysteine protease (ID: 54002) N: 24, σ: 18/24, T: 18.4									
1	HCQS	18	2	34	4	WWGS	18	3	44
2	HCQG	18	3	34	5	WGNS	18	4	44
3	WGSG	18	5	43					
Nuclear receptor ligand-binding domain (ID: 48509) N: 23, σ: 17/23, T: 15.3									
1	FQLL	20	21	43	3	DLQF	17	8	39
2	DLQF	18	7	42	4	LQLL	17	40	31
FAD/NAD-linked reductase (ID: 51943) N : 20 σ: 15/20, T: 90.0									
1	AGGG	17	34	34	2	AGGA	17	91	27

Table 8.1: Structural patterns identified in the Eukaryotic Serine Protease, Papain-like Cysteine Protease, and Nuclear Binding Domains. N: total number of structures included in the data set. σ: The support threshold used to obtain recurring structural patterns, T: processing time (in unit of second). Composition: the sequence of one-letter residue codes for the residue composition of the pattern, κ: the actual support value of a pattern in the family, δ, the background frequency of the pattern, and P: the functional enrichment defined by Equation 8.1. The packing patterns were sorted first by their support values in descending order, and then by their background frequencies in ascending order. The two patterns from FAD/NAD-linked reductase show functional enrichment in NAD(P)-binding Rossman fold protein with -log(P) value 8 and 6, respectively. This is further discussed in Section 8.6.3.

of a family F and no one outside F (i.e. $F = S$) for a large family F, we estimate that this pattern is statistically specifically associated with the family; the statistical significance of the case is measured by a *P–value* close to zero.

We adopt the Bonferroni correction for multiple independent hypotheses [Sha95]: $0.001/|C|$, where $|C|$ is the set of categories, is used as the default threshold to measure the significance of the P-value of individual test. Since the total number of SCOP families is 2327, a good starting point of P-value upper bound is 10^{-7}.

8.6 Case Studies

As a proof-of-concept, we applied the method to identify families specific fingerprints, i.e. structural patterns which occur frequently in a family and rarely outside it. In Table 8.1, a group of four SCOP families are listed which have more than twenty members. This group of families has been well studied in literature and hence comparison of our results with experimental data is feasible.

8.6.1 Eukaryotic Serine Proteases

The structural patterns identified from the ESP family were documented at the top part of Table 8.1. The data indicated that the patterns we found are highly specific to the ESP family, measured by $P\text{–}value \leq 10^{-82}$. We further investigated the spatial distribution of the residues covered by those patterns, by plotting all residues covered by at least one pattern in the structure of a trypsin: 1HJ9, shown in Figure 8.5. Interestingly, as illustrated by this Figure, we found that all these residues are confined to the vicinity of the catalytic triad of 1HJ9, namely: HIS57-ASP102-SER195, confirming a known fact that the geometry of the catalytic triad and its spatially adjacent residues are rigid, which is probably responsible for functional specificity of the enzyme.

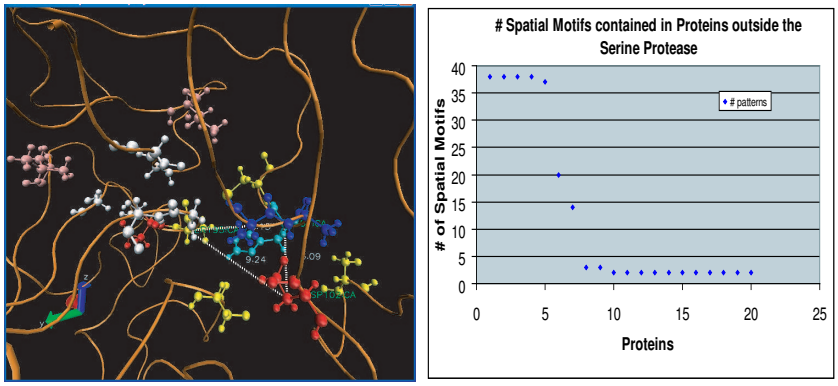

Figure 8.5: Left: Spatial distribution of residues found in 38 common structural patterns within protein 1HJ9. The residues of catalytic triad, HIS57-ASP102-SER195, are connected by white dotted lines. Right: Instances of ESP structural patterns occurring in proteins outside the ESP data set. The the top 7 proteins, where more than ten structural patterns occur, were found to be eukaryotic serine proteases not annotated in SCOP

We found that there are five patterns that occur significantly ($P\text{–}value < 10^{-7}$) in another SCOP family: Prokaryotic Serine Protease (details not shown). This is not surprising since prokaryotic and eukaryotic serine proteases are similar at both structural and functional levels and they share the same SCOP superfamily classification. None of the patterns had significant presence outside these two families.

The SCOP classification (v1.65) used in this paper was released in December 2003. The

submissions to PDB since that time offer a good test of our method to see if we would annotate any new submissions as ESPs. We searched all new submissions for occurrences of the 32 structural patterns we had extracted from the ESP family and find seven proteins: 1pq7a, 1os8a, 1op0a, 1p57b, 1s83a, 1ssxa, and 1md8a, that contain quite a few patterns, as shown in Figure 8.5. All of these proteins are confirmed to be recently published eukaryotic serine proteases as indicated by the headers in corresponding PDB entries.

Finally, we observed that if we randomly sample two proteins from the ESP family and search for common structural patterns, we obtain an average of 2300 patterns per experiment for one thousand runs. Such patterns are characterized by poor statistical significance and are not specific to known functional sites in the ESP. If we require a structural pattern to appear in at least 24 of a 31 randomly selected ESP proteins and repeat the same experiment, we obtain an average of 65 patterns per experiment with much improved statistical significance. This experiment demonstrates that obtaining structural patterns from a group of proteins helps improve the quality of the result, as observed by [WTR$^+$03].

8.6.2 Papain-like Cysteine Protease and Nuclear Binding Domain

We applied our approach to two additional SCOP families: Papain-Like Cysteine Protease (PCP, ID: 54002) and Nuclear Receptor Ligand-Binding Domain (NB, ID: 48509). The results are documented in the middle part of Table 8.1.

For PCP family, we have identified five structural patterns which covered the catalytic CYC-HIS dyad and nearby residues ASN and SER which are known to interact with the dyad [CLA99], as shown in Figure 8.6. For the NB family, we identified four patterns [1] which map to the cofactor binding sites [WBL$^+$03], shown in the same Figure. In addition, four members missed by SCOP: 1srv, 1khq, and 1o0e were identified for the PCP family and six members 1sj0, 1rkg, 1osh, 1nq7, 1pq9, 1nrl were identified for the NB family.

8.6.3 FAD/NAD Binding Proteins

In the SCOP database, there are two superfamilies of NADPH binding proteins, the FAD/NAD(P)-binding domains and the NAD(P)-binding Rossmann-fold domains, which share no sequence or fold similarity. This presents a challenging test case for our system to check whether we are able to find patterns with biological significance across the two groups.

We applied the SM Miner to the largest family in SCOP FAD/NAD(P)-binding domain: FAD/NAD-linked reductases (SCOPID: 51943). With support threshold 15/20, we obtained two recurring structural patterns from the family, and both showed strong statistical signifi-

[1] structural patterns 2 and 3 have the same residue composition but they have different residue contact patterns and therefore regarded as two patterns. They do not map to the same set of residues.

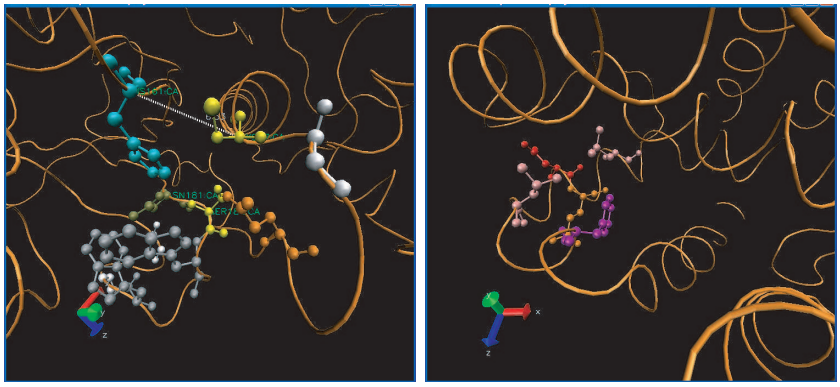

Figure 8.6: Left: Residues included in the patterns from PCP family in protein 1CQD. The residues in catalytic dyad CYS27-HIS161 are connected by a white dotted line and two important surrounding residues ASN181 and SER182 are labeled. Right: Residues included in patterns from the NB family in protein 1OVL. The labeled residue GLN 435 has direct interaction with the cofactor of the protein.

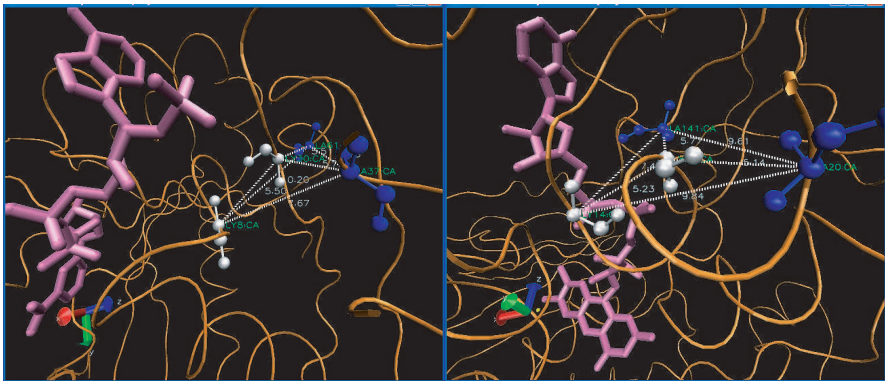

Figure 8.7:]
The pattern appears in two proteins 1LVL (belongs to the FAD/NAD-linked reducatase family without Rossman fold) and 1JAY (belongs to the 6-phosphogluconate dehydrogenase-like, N-terminal domain family with Rossman fold) with conserved geometry.

cance in the NAD(P)-binding Rossmann-fold superfamily as shown in bottom part of Table 8.1.

In Figure 8.7, we show a pattern that is statistically enriched in both families; it has conserved geometry and is interacting with the NADPH molecule in two proteins belonging to the two families. Notice that we do not include any information from NADPH molecule during our search, and we identified this pattern due to its strong structural conservation among proteins in a SCOP superfamily. The two proteins have only 16% sequence similarity and adopt different folds (DALI z-score 4.5). The result suggest that significantly common features can be inferred from proteins with no apparent sequence and fold similarity.

Chapter 9

Function Annotation of Proteins with Structure Patterns

9.1 Introduction

Structural genomics projects [Bur00] have generated many structures for proteins with unknown function; these are called hypothetical proteins. Traditionally the function of hypothetical proteins are inferred by recognizing similarities, at the sequence or global structural level, with proteins of known function. Though successful in the past, these approaches have limited power for structural genomics proteins since those structures are carefully selected to avoid strong sequence and overall structure similarity to known proteins in order to sample the overall protein "structure space". Quantitative study revealed that significant proportion of all structural genomics proteins ($\sim 25\%$) have sequence identity less than 30% and z-score less than 10 according to DALI [HS96] with all proteins of known function [**?**]; such proteins are often called *orphan* structures.

Recently a new approach has been developed to infer the function of orphan structures that does not rely on sequence and overall structure similarity. In this method, protein function was inferred by finding similarities in the geometric arrangement of residues in the functional site with that in proteins of known function. For example, Aloy *et al.* found that conserved geometric packing patterns of a few residues are often responsible for protein function, and can lead to more accurate function inference than by global structural homology [AQAS01]. Laskowski *et al.* developed the reverse template method of SiteSeer [LWT05b], which searches for all three-residue structural fragments within a protein in the entire PDB, and then checks the alignment of a sphere of residues around the matched ones. There are also methods using Gene Ontology [Con04] as a reference to define function, such as ProKnow [PE05] and PHUNCTIONER [PS04].

Graph representations of protein structure allow more flexibility than templates in representing and matching structural motifs. Earlier methods use graph representations of protein

structure to search for known patterns [APG+94, SR03], or determine patterns with limited topology, such as cliques, from groups of proteins [WTR+03, MSO03]. Other recent methods find functionally important residues using computed chemical properties [KMWO05], careful alignments [PBO+05], and evolutionary information [WY05].

In our previous chapters, we defined a graph representation using the *almost-Delaunay edges* [BS04] for proximity [HWB+04, HBW+05]. Demonstrated by several case studies such as the serine protease and kinase families, we showed that structure patterns often have biological relevance, cover functionally important residues, and can be used to accurately classify proteins from similar families [HBW+05].

In this chapter, we present a method for function inference that utilizes our pattern discovery method. The method searches for structure patterns using previously developed method, and assigns a significance score to family membership using the distribution of patterns found in members of the family and in the background.

This method assumes no knowledge of functional sites, and can find patterns of arbitrary shape. Our pattern is statistically specific to its family, and our consensus approach using multiple patterns improves the accuracy and specificity of function inference. Families with different function and similar structure can be distinguished, since the patterns tend to identify functionally important parts of a protein automatically. Whereas current methods using the Gene Ontology work better at inferring broader functional categories than specific functional families [PS04, PE05], the strength of our method is in distinguishing among specific, related and similar functions.

Since our structure pattern are usually statically specific to a certain protein structure family, below, we also call a structure pattern a structure *fingerprint*.

9.2 Methods Overview

We now describe the steps of the fingerprint-based function inference algorithm in detail.

9.2.1 Family and background selection

We selected 120 families and superfamilies from SCOP version 1.65, the current version when we began this work. Though SCOP 1.67 was released in February 2005, we have retained the fingerprints derived from SCOP 1.65 to allow unbiased function prediction of structural orphans using information known at the time they were selected, and use new members added in SCOP 1.67 to validate the method. Non-redundancy of the family members was ensured by intersection with dataset selected using PISCES [WD03] with 90% sequence identity, better than 3 Å resolution, and R-factor at most 1.0. These criteria generated a representative set of 6749 protein chains in May 2005, that we used as the *background* when identifying fingerprints.

The lists of families selected, members of each family and the background dataset are listed in supplementary data on our project website: http://www.cs.unc.edu/~debug/papers/FuncInf

9.2.2 Graph Representation

We represent protein structures as graphs, with nodes at each residue labeled with the amino acid type. Amino acids V, A, I, L are condensed to a single residue type since they are frequently substituted for one another, while the other residue types are distinct. Contact between residues is defined using the *almost-Delaunay* edges [BS04]. In the *simple edge* representation, these are the only criteria. In the *distance edge* representation, we add distinct labels for edge length ranges (0–4,4–6, 6–8.5, 8.5–10.5, 10.5–12.5 and 12.5–15 Å, loosely based on statistics of inter-C_α distances in proteins), and distance constraint edges between non-contacting residues to ensure consistent geometry in patterns found. Since function inference results are generally found to be superior using fingerprints derived from distance edges [BS04], we use that representation in this paper. The exception is the metallo-dependent hydrolase family that has strong simple edge fingerprints but weak distance edge fingerprints; we give results from simple edge fingerprints for this family, and are investigating why the distance edge fingerprints are weak.

9.2.3 Frequent Subgraph Mining

We mine frequent subgraphs from the graph representation of all proteins in a family using the Fast Frequent Subgraph Mining method[HBW$^+$05]. We use a support value of 80% to define frequency. Frequent subgraphs are constrained to have high density by having no more than one edge missing from a clique.

Fingerprints are defined as those frequent subgraphs that occur in at least 80% of the family (support), and at most 5% of the background (background occurrence). Typical families in our dataset have 10–1000 fingerprints, and the support and background occurrence are adjusted for small or heterogeneous families until the target number of fingerprints is attained.

9.2.4 Search for Fingerprints in a Query Protein

We use an index of graph similarity to speed up Ullman's subgraph isomorphism algorithm [Ull76]. We create an index vector for each node of the fingerprints and the query structure that stores the node and edge labels of all nodes connected to it. Each potential embedding in the query of a node in a fingerprint is considered only if the index vectors match. This reduces billions of potential embeddings of the fingerprint to a handful in most cases. Ullman's algorithm then finds all consistent embeddings of the fingerprint in the query, with the same node and edge labels as in the fingerprint.

9.2.5 Assigning Significance

We assign significance to the function inference by examining the number of fingerprints found against the distribution of fingerprints in background proteins and in family members. These distributions are not normal, and hence we use an empirical p-value calculation rather than a parametric one. If we pick different numbers of fingerprints at which to infer family membership, we can determine the rate of true and false positives and negatives, calculate specificity and sensitivity, and draw ROC curves as shown in the inset of Figure 9.1(a)–(b). We choose two cutoff points for each family: a *sensitivity cutoff* to maximize sensitivity with at least 95% specificity, and a higher *99%-specificity cutoff* with no constraints on sensitivity.

As outlined in the previous section, we use the FFSM subgraph mining program, available at (http://www.cs.unc.edu/~huan/FFSM/) to identify patterns in protein structures and the ADMATLAB package, available at http://www.cs.unc.edu/~debug/software/ to evaluate the significance of the structure patterns.

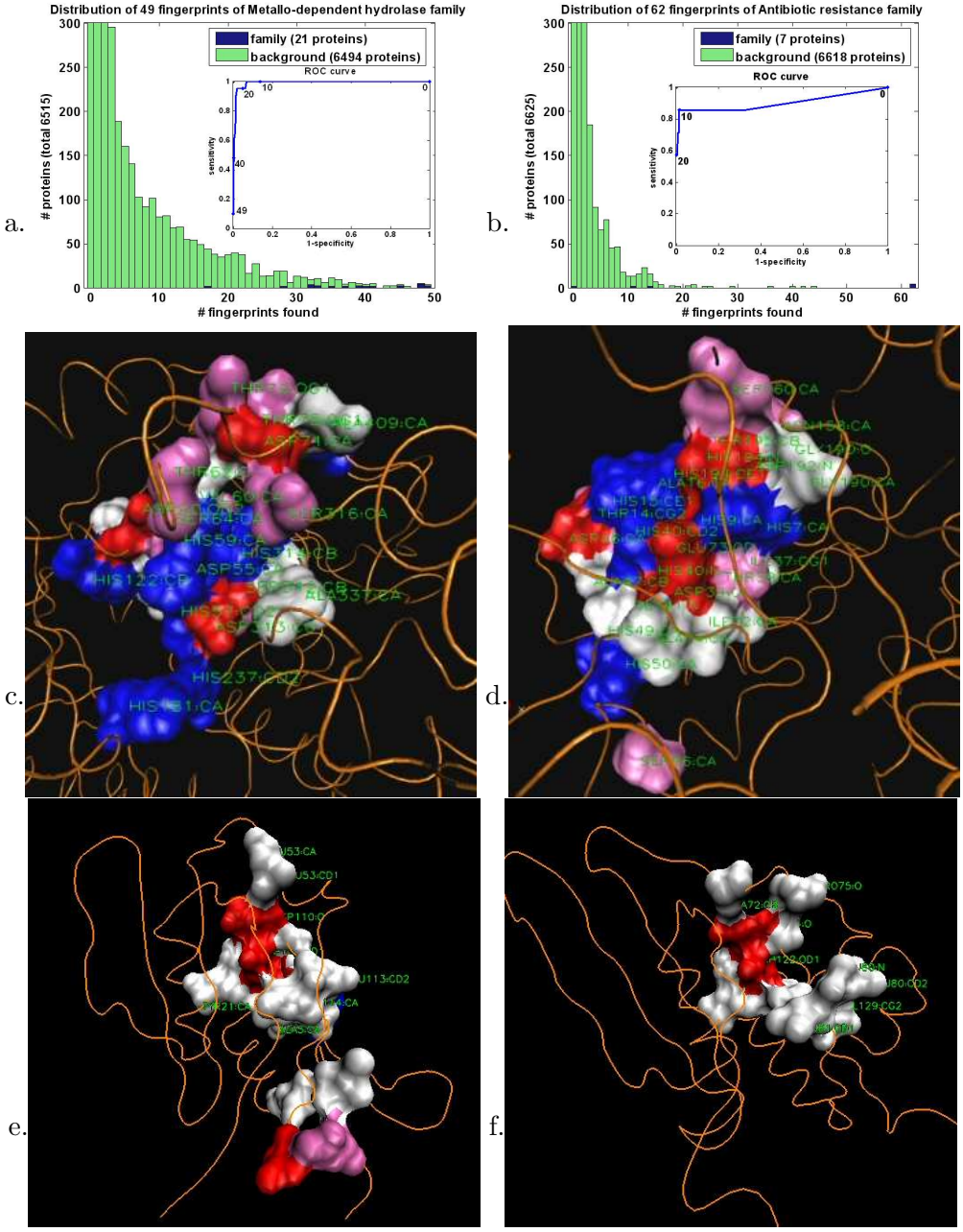

Figure 9.1: Distribution of (a) Metallo-dependent hydrolase (SCOP: 51556) and (b) Antibiotic resistance (SCOP: 54598) fingerprints in the background (light bars), and within the family (dark). Inset: ROC curve showing specificity vs. sensitivity of function inference at different numbers of fingerprints. (c)–(d) Example of function inference: residues covered by metallo-dependent hydrolase fingerprints in (c) the metallo-dependent hydrolase 1nfg, and (d) 1m65 (Ycdx, unknown function). (e)–(f) Another example of function inference: residues covered by antibiotic resistance fingerprints in (e) the family protein 1ecs, and (f) 1twu (Yyce, unknown function). Figures (c)–(f) are snapshots from VMD [HDS96]. See also the kinemages in the online supplementary material.

(a)

abrv	SCOP (sup)fam	#mem	#fp	$\#fp_{max}^{TIMfold}$
ALD	Aldolase (Class I)	17	74	41 (1gteA=FMN)
AKN	Aldo-keto reductase	9	72	4 (1hvxA=AMY)
AMY	Amylase	39	42	23 (1mucA=DGL)
BLU	Bacterial luciferase	6	86	20 (1ofdA=FMN)
BGL	Beta-glycanase	27	28	19 (1cgt=AMY)
B12	Cobalamin (B12) dependent enzyme	5	66	8 (1a3wA=PYK)
DGL	D-glucarate dehydratase-like	9	29	6 (1gkrA=MDH)
DHP	Dihydropteroate synthase-like	4	41	5 (1eepA=IMP)
ENC	Enolase C-terminal domain like	14	111	60 (1ofdA=FMN)
FMN	FMN-linked oxidoreductase	14	55	31 (1me8A=IMP)
GH1	Glycosyl hydrolase family 1	9	887	12 (1bqcA=BGL)
HIS	Histidine biosynthesis enzyme	5	290	47 (1ofdA=FMN)
MDH	Metallo-dependent hydrolase	17	20	14 (1bxnA=RBC)
PYK	Pyruvate kinase	5	819	25 (1ea0A=FMN)
RIB	Ribulose-phosphate binding barrel	17	20	19 (1jr1A=IMP)
RBC	RuBisCo	7	315	14 (1ea0A=FMN)
TIM	Triosephosphate isomerase	12	366	9 (1ea0A=FMN)
TRP	Tryptophan biosynthesis enzyme	6	22	15 (1pvnA=IMP)
XIM	Xylose isomerase	7	758	10 (1n8iA=MAL)
XIL	Xylose-isomerase-like	11	17	5 (1e0tA=PYK)

(b)

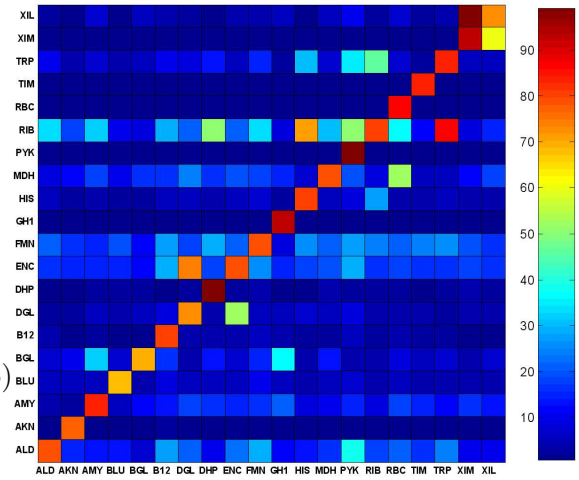

Figure 9.2: Discriminating the TIM barrels using fingerprints. (a) The 20 families selected, with columns listing a 3-letter abbreviation for each family, number of members and fingerprints, and maximum number of fingerprints found in a non-family protein of the TIM fold. Families mentioned in the last column for which fingerprints were not identified: IMP(inosine monophosphate dehydrogenase) and MAL (malate synthase). (b) Pseudo-color matrix plot showing the percentage of fingerprints of the TIM barrel family in each row found in an average member of the family in each column. High values on the diagonal (red) and low off-diagonal (blue) indicate high discrimination. Exceptions to this trend are documented in the text.

9.3 Results

We examined the family specificity of the fingerprints, validated the method by identifying cases of functional similarity with/without overall structure similarity, and applied it to function inference of orphan structures from structural genomics.

9.3.1 Fingerprint Occurrence in Family and Background

Examining the number of fingerprints of different families found in the non-redundant background dataset helps test the uniqueness of a family's fingerprints, and establish significance of a function inference, as described later in Materials and Methods. For most families, most background proteins have fewer fingerprints than the minimum found in any family member; see Figure 9.1(a)–(b) for examples of the metallo-dependent hydrolase (SCOP: 51556) and antibiotic resistance (SCOP:54598) families. Some family members have few fingerprints; usually they have a different function or mechanism from the other members, or have errors in the structure file that prevent the identification of fingerprints.

Background proteins that have most of the fingerprints turn out to be new family members or proteins from other families with the same function. For example, four proteins with 30 or more of the 49 metallo-dependent hydrolase fingerprints: 1un7A (48), 1rk6A (40), 1ndyA (33) and 1kcxA (32), were not included in the SCOP 1.65 family from which the fingerprints were determined, but are classified in the SCOP 1.67 family. Other high-scoring proteins have closely related functions such as metallohydrolase/oxidoreductase of TIM barrel fold (1p9e, 48), mannose hydrolase of the $(\beta\alpha)_7$ fold (1qwn, 44), and other functions involving metal-ion coordination and phosphate hydrolysis.

9.3.2 Validation on SCOP

SCOP superfamilies and families are known to be evolutionarily (and usually functionally) related. Thus, to test the validity of our method, we used the fingerprints from SCOP 1.65 to classify proteins newly added to these families in SCOP 1.67. The detailed results are shown in Tables 9.1– 9.3 in the online supplementary material. Of the 515 new members added to 105 families, the function of 385 (75%) was inferred using the fingerprints from the SCOP 1.65 family at the sensitivity cutoff, and of 351 (68%) at the 99%-specificity cutoff. 69% of all newly added members are inferred with higher specificity than all inferences for competing families in the current dataset, and with at least 95% specificity. The last number is most important, since among several competing function inferences one is likely to select the one with highest specificity.

SCOP ID	SCOP (super)family	# prot	# fp	new family members inferred #	sens	spec	other >spec	other fns. inferred spec	sens
53384	AAT-like	14	36	2	2	2	0	1	5
52686	ABC transporter ATPase domain-like	12	17	3	3	1	2	6	10
55753	Actin depolymerizing protein	9	94	1	1	1	0	2	3
52402	Adenine nucleotide α-hydrolase	16	13	3	2	0	1	1	3
51431	Aldo-keto reductases (NADP)	9	72	8	8	8	0	2	9
53649	Alkaline phosphatase-like	7	75	2	1	1	0	2	6
51446	Amylase, catalytic domain	39	42	3	2	2	0	3	8
54598	Antibiotic resistance protein	4	62	3	2	1	1	1	1
48371	ARM repeat	18	294	6	2	1	1	1	2
53570	Bacterial lipase	5	132	5	5	5	0	6	9
53057	β-carbonic anhydrase	4	62	4	4	4	0	2	7
51487	β-glycanase	27	152	9	7	5	1	3	9
51487	β-glycanase	27	28	9	6	5	0	2	5
56602	β-Lactamase/D-ala carboxypeptidase	20	73	7	5	5	1	4	10
48727	Antibody variable (V) domain	310	92	63	63	63	1	1	3
53487	Carboxylesterase	5	59	1	1	1	0	2	8
52243	Cobalamin (vitamin B12)-binding	4	35	4	4	4	0	21	37
75434	CutA divalent ion tolerance protein	3	58	4	4	4	0	1	2
49330	Cu,Zn superoxide dismutase-like	10	183	1	1	1	0	1	1
50353	Cytokine	9	45	4	4	3	0	1	1
51609	D-glucarate dehydratase-like	9	29	3	3	3	0	6	12
52467	DHS-like NAD/FAD-binding domain	14	31	5	3	2	3	27	38
51717	Dihydropteroate synthetase-like	4	41	1	1	1	0	10	17
53118	DnaQ-like 3'-5' exonuclease	11	15	3	2	2	1	10	23
57196	EGF/Laminin	21	24	3	3	3	0	3	4
51604	Enolase C-terminal domain-like	14	111	3	2	1	1	5	8
52432	ETFP subunit	6	85	6	6	6	0	11	22
50514	Eukaryotic serine protease	56	155	5	4	4	0	3	5
81269	Extended AAA-ATPase domain	17	13	6	4	4	2	18	27
54602	Extradiol dioxygenase	6	150	6	6	6	0	6	8
51521	Glycosyl hydrolase Family 1	9	887	9	9	9	0	15	22
46610	Fe,Mn superoxide dismutase (SOD)	14	113	2	2	2	0	2	3
50354	Fibroblast growth factor (FGF)	6	143	3	3	3	0	1	1
51396	FMN-linked oxidoreductase	14	55	8	4	4	1	21	35
53558	Fungal lipase	8	85	1	1	1	0	1	2
52592	G protein	42	39	8	8	8	0	10	17
52318	Glutamine amidotransferase Class I	10	55	2	2	0	1	1	2
55931	Glutamine synthetase/guanido kinase	8	16	1	1	1	0	1	3
51011	Glycosyl hydrolase domain	37	81	6	2	2	0	3	10
56784	HAD-like	9	45	10	9	7	1	8	13
51367	Histidine biosynthesis enzyme	5	290	5	5	5	0	6	12
49944	Laminin G-like module	4	119	1	1	0	1	1	1
51556	Metallo-dependent hydrolase	17	20	4	3	1	1	9	19
56281	Metallo-hydrolase/oxidoreductase	7	30	2	1	1	0	0	1
55486	Metalloprotease ("zincin")	42	34	5	5	3	1	3	4
52641	Motor protein	11	25	4	3	3	0	1	6
49550	Multidomain cupredoxin	11	222	1	1	1	0	2	4
52403	N-type ATP pyrophosphatase	6	118	2	1	1	0	3	4
48509	Nuclear receptor ligand-binding domain	23	83	7	7	7	0	3	5
55811	Nudix	5	211	7	5	5	0	1	3
49417	p53-like transcription factor	11	57	3	1	1	0	0	0
54002	Papain-like	19	178	9	7	7	0	1	3
50646	Pepsin-like acid protease	24	145	3	3	3	0	3	5
48537	Phospholipase C/P1 nuclease	4	32	1	1	1	0	1	2
50495	Prokaryotic serine protease	10	11	4	3	3	2	2	4
54815	Prokaryotic type KH domain (type II)	5	76	5	5	5	0	19	31
88854	Protein kinases, catalytic subunit	32	92	17	17	17	1	5	10
53167	Purine and uridine phosphorylase	7	129	5	1	1	0	0	2
52475	Pyruvate oxidase and decarboxylase	5	177	3	3	3	1	33	45
52670	RecA protein-like (ATPase-domain)	11	15	2	1	1	1	1	3
50371	Ricin B-like	7	412	3	3	3	0	2	6
56575	Serpins	12	287	6	1	1	0	0	0
53697	SIS domain	6	453	2	1	1	0	0	2
50386	STI-like	7	91	7	7	7	0	20	27
52744	Subtilase	7	53	4	4	4	1	21	35
51381	Tryptophan biosynthesis enzyme	6	22	3	2	0	1	2	10
53187	Zn-dependent exopeptidase	11	14	13	3	1	2	3	4

Table 9.1: Function inference of new members in SCOP 1.67 using distance edge fingerprints (only metallo-dependent hydrolase shown with simple edge fingerprints). The left part of the table describes the family used for mining fingerprints; the middle part lists the number of new members added, the number inferred at the sensitivity and 99% specificity cutoffs and the number inferred more strongly by another family's fingerprints; the right part list the number of other functions inferred for the new members at 99%-specificity and sensitivity cutoffs.

SCOP ID	SCOP (super)family	# prot	# fp	# new members	# fingerprints in new members	sens. cutoff
48179	6-phosphogluconate dehydrogenase C-terminal domain-like	12	16	1	5	9
51570	Aldolase, class I	17	74	6	20 19 17 15 11 7	
75217	α/β knot	6	22	6	2 2 2 1 1 0	3
51883	Aminoacid dehydrogenase-like, C-terminal domain	18	376	3	111 100 64	145
51679	Bacterial luciferase-like	6	86	2	6 5	8
56655	Carbohydrate phosphatase	12	69	1	16	21
49384	Carbohydrate-binding domain	7	17	1	3	4
49348	Clathrin adaptor appendage domain	5	77	1	4	15
53402	Cystathionine synthase-like	15	17	3	3 3 0	6
56236	Glutamine amidotransferase Class II	7	83	1	9	37
55931	Glutamine synthetase/guanido kinase	8	941	1	3	9
54060	His-Me finger endonucleases	5	57	1	11	18
55608	Homing endonucleases	4	25	1	6	7
53659	Isocitrate/Isopropylmalate dehydrogenase-like	8	208	3	16 15 7	17
63379	MarR-like transcriptional regulators	5	36	2	5 0	16
53218	Molybdenum cofactor biosynthesis proteins	5	73	1	1	6
81301	Nucleotidyltransferase	6	110	5	24 17 17 16 6	29
49354	PapD-like	7	123	1	9	27
53822	Periplasmic binding protein-like I	8	23	6	6 3 2 2 1 0	9
88723	PIN domain-like	6	78	1	19	24
53099	Ribonuclease H	7	155	1	8	24
51366	Ribulose-phoshate binding barrel	17	20	4	5 3 1 1	8
46928	RuvA C-terminal domain-like	7	59	2	6 5	23
55620	Tetrahydrobiopterin biosynthesis enzymes	4	27	3	5 1 0	9
48453	Tetratricopeptide repeat (TPR)	9	14	1	2	6
54637	Thioesterase/thiol ester dehydrase-isomerase	7	35	8	7 6 5 5 4 3 3 3	15

Table 9.2: The families from Table 9.1 for which none of the new members were inferred; the number of fingerprints in each new member and the sensitivity cutoff points are also shown.

SCOP ID	SCOP (super)family	# prot	# fp	new family members inferred #	sens	spec	other >spec	other fns. inferred spec	sens
50090	Electron transport accessory protein	5	28	5	5	5	0	0	0
50157	PDZ domain	9	132	5	5	5	0	0	0
63550	Penta-EF-hand protein	4	107	4	4	4	0	0	0
52266	SGNH hydrolase	5	42	1	1	1	0	0	0
54496	Ubiquitin conjugating enzyme	14	19	2	2	2	0	0	0
48468	VHS domain	4	84	1	1	1	0	0	0
57668	Zinc finger, classic C2H2	5	17	1	1	1	0	0	0
48942	*Antibody constant (C1) domains*	259	12	*57*	*57*	*49*	*0*	*0*	2
56317	*Carbon-nitrogen hydrolase*	3	97	*1*	*1*	*1*	*0*	*0*	2
53301	*Integrin A (or I) domain*	6	214	*3*	*3*	*2*	*0*	*0*	2
56300	*Metallo-dependent phosphatase*	10	316	*1*	*1*	*1*	*0*	*0*	3
50045	*SH3-domain*	17	17	*4*	*4*	*4*	*0*	*0*	1
55299	*YjgF/L-PSP*	4	131	3	3	3	0	0	2

Table 9.3: The families from Table 9.1 for which all new members added in SCOP 1.67 were inferred using fingerprints of the SCOP 1.65 family, and there were either no other functions inferred (top half of tables) or there were no other functions inferred with 99% specificity or with higher specificity than the family fingerprints (bottom half, italics).

9.3.3 Discrimination between similar structures with different function

To test the discrimination power of fingerprints, we searched for the fingerprints of 20 structurally similar (super)families of the TIM barrel fold that have different functions. As shown in Figure 9.2, the average member of any of these families has 70–90% of the fingerprints of its own family (orange or red, seen on the diagonal), and 0–40% of the fingerprints of any other family (blue, seen off the diagonal). Exceptions arise from superfamily-subfamily pairs such as enolase C-terminal domains(ENC) and D-glucarate dehydratases(DGL) that share fingerprints since their members overlap, and from families with weak fingerprints that cannot distinguish it from similar families, such as the ribulose-phosphate-binding barrels(RIB). Overall, the method is shown to discriminate between functional families whose members cannot be distinguished easily by overall structural similarity.

9.3.4 Function inference for structural genomics proteins

Next, we classified Structural Genomics targets in the PDB as proteins with known function, putative function suggested by overall structure similarity, and no function known or suggested by overall similarity. Our method suggested function assignments for proteins in the last two categories. For example, functional similarity of the protein 1npy to the shikimate dehydrogenase family (sequence identity 22% 1npd) was inferred from overall structural similarity and confirmed by finding 33 of 114 family fingerprints, which implies 100% specificity for function inference.

In other cases, there was no detectable sequence or structural similarity. Here we present two case studies on metallo-dependent hydrolases and antibiotic resistance proteins.

Functional Inference of YcdX protein

The YcdX protein (PDB: 1m65, CASP5 target T0147) has a rare $(\beta\alpha)_7$ barrel fold called the PHP domain (SCOP: 89551), and had no significant sequence or overall structure similarity with proteins of known function in 2004. We inferred that this protein has a metallo-dependent hydrolase function, with 30 of 49 fingerprints from SCOP superfamily 51556, a TIM barrel family. After fingerprint-based inference, we also visualized the residues included in fingerprints within proteins of the family and the target protein, to verify the biological significance of the local structures matched by fingerprints. These residues are localized in space and show similar geometric arrangements and chemical properties in family and target, as shown in Figure 9.1(c)–(d).

Our inference was validated by (1) active site template and reverse template matches on the ProFunc server [LWT05a, LWT05b], (2) suggestions by the CASP5 target classifiers[KQHG03], and (3) suggestions by the authors of the structure [TOK$^+$03]. The PINTS-weekly service [SSR04] found active site patterns from many metallo-dependent hydrolases in this protein.

Finally, GenProtEC, the E.Coli genome and proteome database [SGR04] has annotated the YcdX gene product as belonging to the SCOP metallo-dependent hydrolase structural domain family, on the basis of the SUPERFAMILY database of HMMs for SCOP families [GC02, MVK+04].

Functional Inference for Protein Yyce

Protein Yyce from Bacillus Subtilis (PDB: 1twu) is unclassified in both SCOP 1.65 and 1.67, and was an orphan structure in 2004, with no structural similarity to structures of known function. We found 46 of 62 fingerprints from the antibiotic resistance protein family (SCOP ID: 54598) in 1twu, inferring the antibiotic resistance function with specificity 100%. Figure 9.1(e)–(f) show the residues covered by fingerprints in 1ecs, an antibiotic resistance protein in SCOP 1.65, and in 1twu. Note the geometric and electrostatic similarity between the upper region covered by fingerprints in 1ecs and the one in 1twu.

Subsequently, when structural similarity of protein 1twu was re-evaluated in May 2005 using the current DALI database, it was found to be similar to a protein 1nki that was unclassified in SCOP 1.65 but has been added to the antibiotic resistance protein family in SCOP 1.67. This discovery of homology to a newly classified member of the family confirms our function inference.

Several other strong function inferences for orphan structures and hypothetical proteins have yet to be verified; they will be presented in an upcoming full-length paper on this work.

9.4 Discussion

We present a method to infer a specific function for proteins of unknown function, independent of sequence patterns, structural alignments and templates of known functional sites, which complements the other methods and may succeed where they fail. The use of multiple motifs mined from protein families using almost-Delaunay graph edges makes the method especially robust. Our method can find and identify packing patterns of functionally important residues even when there are distortions in the functional site. Unlike clique patterns [WTR+03, MSO03] and triplets [BKB02, LWT05b], our method finds family-specific patterns of arbitrary topology and size.

The successful inference of new members of SCOP families validates the predictive power of fingerprints and the decision to use families from a structural classification; the success rate of 69% is high considering that there are functional outliers among existing and new members of SCOP families. The packing patterns in fingerprints do capture functional information that is unique to a functional family, rather than shared structural information. This is shown by function discrimination within the TIM barrel fold, and the inference of YcdX as belonging to the sequence-diverse metallo-dependent hydrolase family even though it has a different fold.

Our method can be trained on families from a functional classification system such as EC [Bai00] or GO [Con04], and we will report these results in the future.

Our structure-based method may be applied to function prediction at the sequence level using either good quality predicted structures, or sequence patterns derived from fingerprints whose sequence order is preserved within the family. We have preliminary results on this that will be reported in a future publication.

We do observe annotations that oppose our inference. For example, the Gene Ontology Annotation (GOA) database [Con04] annotates 1m65 (which we believe is a metallo-dependent hydrolase) as having *DNA-directed DNA polymerase activity* (GO:0003887). We notice that this function assignment is putative and is made on the basis of electronic annotation transferred from InterPro, a sequence database. This is the least reliable evidence code for GO annotation, Inferred from Electronic Annotation(IEA). The discoverers of the PHP domain sequence family [AK98] indicated shared active site sequence motifs between the metallo-dependent hydrolase family and the PHP-domain family, and hypothesized that bacterial and archaeal DNA polymerases possess intrinsic phosphatase activity that hydrolyzes the pyrophosphate released during nucleotide polymerization. Several metallo-dependent hydrolases hydrolyze phosphoester or phosphate bonds; thus, the assigned GO term may not contradict the function inferred by our method, which is left for further study.

Our method has a few limitations, arising from implementation choices, algorithmic issues, and the nature of the problem itself. In our implementation, we use C_α coordinates to calculate graph edges and lengths; this choice captures shared topology, but may miss interactions with long side-chains. Currently we do not allow residue substitutions in patterns, other than unifying V,A,I,L. Merging commonly substituted residue types (e.g. D,E) increases the sensitivity of fingerprints but can decrease their specificity; we may lose fingerprints that are no longer unique to a family. Also, the distance edge matching criteria may be too restrictive to find some patterns that vary widely in their geometry or have distances lying on bin boundaries. We are developing a new distance edge representation that will remedy this.

Algorithmically, subgraph mining is an NP-complete problem since it involves subgraph isomorphism. Though the FFSM algorithm [HWB$^+$04] avoids the isomorphism problem in most cases by storing graph embeddings, it does affect families with very similar or identical structures, and families of 3 or fewer proteins. Such families often yield excessive and non-specific fingerprints that have little power to infer new members. Also, finding fingerprints in a family is not guaranteed; roughly 35% of SCOP families that we tried gave no fingerprints (typically functionally heterogeneous SCOP superfamilies, and some families with extreme sequence divergence) or too many fingerprints (typically highly sequence-similar enzyme families). The number, specificity and sensitivity of fingerprints depends on family characteristics such as size and heterogeneity; thus the support and background occurrence parameters must be varied to find meaningful sets of fingerprints in the maximum number of families.

On the systemic level, the method identifies functional families with four or more representatives by learning what packing patterns are unique to each family, and it does not identify the exact function within a family. Also, several structures have errors, missing fragments or mutations that lead to failure of mining or function inference. Careful manual selection of families and fixing errors in structures should improve the results further. Finally, all function inferences made by computational methods such as ours need to be validated in the laboratory.

Chapter 10

Coordinated Evolution of Protein Sequences and Structures

10.1 Introduction

Evolutionary relationships between extant proteins afford a valuable database with which to systematically examine how protein structures derive from their sequences. Protein families exhibit immense population diversity in which many families contain only a few members while a few families contain large numbers of related structures. Large protein families include the TIM barrel family and the Rossman fold family among others, which account for 30% of the entire proteome [SMM+11]. An important key to understanding the origins of such large families may be found by identifying sequence patterns and structure patterns that are associated with such large fold classes. The study of the relation between protein sequence spaces and protein structure spaces have reached a significant plateau, from which one can envision important new insights from an appropriate analysis of the joint distributions of conserved sequence and structure motifs [KWK04].

Our investigation springs from previous studies of protein fold families, which comprise collections of proteins whose structures are similar, but whose sequences differ considerably [KWK04]. In those fold families, conserved sequence sites have been studied extensively. For example, Dokholyan *et al.* defined a measure called sequence entropy to identify conserved sites in protein superfamilies. The concept of sequence entropy was also elaborated by Donald *et al.*, who established a database of Conservatism of Conservatism (CoC) to record the sequence entropy of all residues of many protein structures [DHR+ar]. As reported by Donald *et al.* , residues with low sequence entropy and low solvent accessibility are responsible for rapid folding and thus candidates for a "folding nucleus".

Tertiary structure information is utilized sparingly in most of the current studies for protein fold family evolution. The recent exponential growth in the structure databases has brought with it considerable new potential for knowledge-based methods to study structure

conservation within a fold family. Quantifying and identifying structurally conserved regions can supplement our understanding of protein sequence conservation, enhancing our understanding of how sequence determines structure by identifying correlations between sequence and structure conservation.

To achieve a quantitative measure of structure conservation, we have developed structure entropy to measures the conservation of the local environmental in a protein fold family. Our methods make it feasible to directly relate the sequence and structure conservation during protein evolution [DHR$^+$ar, SDDS03]. Specifically, we transform a protein structure to a consistent contact graph where a node corresponds to an amino acid residue in the protein and an edge represents a pairwise interactions of residues. Given a group of graphs and a node v, we search a subset of nodes that (almost) always connect to v in the entire set of graphs and define structure entropies accordingly. The computation of structure entropy involves a search in a high dimensional space and has been shown to be reasonably stable in the presence of structure noises.

Our result shows that there are strong correlation between the sequence entropy, structure entropy, and frequent structure patterns in a fold family. These result provides the first direct evidence at the residue level for the correlated sequence and structure evolution in protein fold families.

10.2 Methods

We start this section by listing the data set that we study. We then define labeled undirected graphs, which set up our formal framework for describing a protein structure with a graph. After discussing labeled graphs, we present a way to apply labeled graphs to represent a protein structure, known as the protein contact map. Such a graph representation is critical for the subsequent analysis and is hence discussed in detail. With contact maps, we define a novel structure entropy measure quantifying the structure conservation in a fold family. At the end of the section, we introduce the frequent subgraph mining technique and show how it may be applied to locate structure patterns in protein fold families. Before our technical discussions we summarize notations that are used in this section.

10.2.1 Notations

S	a fold family	$N_I^P\|_U$	projected neighbor set of I in P
S_i	a structure in S	$H(Y)$	entropy of random variable Y
I, J	sites in a fold family	$H(I)\|_U$	projected structure entropy of I
N_I	neighborhood of I	$\mathcal{E}_I$	sequence entropy of I
U	a subset of N_I	ξ_I	structure entropy of I
N_I^P	neighbor set of I in a structure P		

10.2.2 Data Sets

We collect five fold families: Immunoglobulin fold (Ig), oligonucleotide-binding fold (OB), Rossman fold (R), α/β-plait($\alpha/\beta - P$), and TIM-barrel fold (TIM). For each fold family, we designate one protein as its representative as done in [DS01]. The data set is summarized in Table 10.1

To estimate the sequence and structure conservation across a large group of structurally-related proteins, we introduce the following distinctions. For a protein P, its *homologs* are proteins that have similar structure to P and have at least 25% (pair-wise) sequence identity to the protein. The *analogs* of P are proteins that have similar structures to P but have less than 25% (pair-wise) sequence identity.

Fold	Ig	OB	R	$\alpha/\beta - P$	TIM
N	205	76	626	1129	193
$PDBID$	1TEN	1MJC	3CHY	2ACY	2EBN
L	89	69	128	98	285

Table 10.1: The data set we use. N is the number of analogs the protein has in the FSSP database. PDB ID is the PDB ID of the representative protein in a fold family and L is the length of the representative protein.

10.2.3 Graph Representation of Proteins

In order to represent protein structures by a rigorous mathematic model, in the sequel we define labeled graphs. We then present our procedure of using graphs to represent protein structures.

Labeled Graphs

Definition 10.2.1 *A **labeled graph** G is a four-element tuple $G = (V, E, \Sigma, \lambda)$ where*

- *V is a set of nodes,*
- *$E \subset V \times V$ is a set of undirected edges joining two distinct nodes,*
- *Σ is the set of node labels and edge labels,*
- *$\lambda : V \cup E \to \Sigma$ maps nodes and edges to their labels.*

Given a graph G the *size* of a graph G is the cardinality of the node set. The *degree* of a node v in G is the number of edges incident with v. A *graph database* is simply a group of labeled graphs. A graph is (node) *unique labeled* if each node has a unique label. As we see from the way that we build graph representations for protein structures, all graphs we deal with are unique labeled graphs.

As an example in Figure 10.1, we show a graph database with three labeled graphs. The labels of nodes and edges are specified within each node and along each edge for each graph. Notice that all three graphs are uniquely labeled since there are no duplicated node labels in any of these graphs.

Since graphs in this chapter always correspond to protein structures, in the subsequent discussion, we treat the two terms graphs and structures interchangeably.

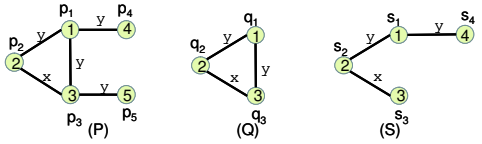

Figure 10.1: A database $\mathcal{G}$ of three labeled graphs.

10.2.4 Indexing Residues according to FSSP

For each representative proteins, we retrieve the analogs of the protein from the FSSP database (**F**old classification based on **S**tructure-**S**tructure alignment of **P**roteins) accessible at http://www.bioinfo.biocenter.helsinki.fi:8080/dali/index.html. The representative protein and its analogs form a *fold family*. We use $S = \{S_1, S_2, \ldots, S_n\}$ denoting an n-protein fold family S with the assumption that S_1 is the representative protein in S and the rest are analogs of S_1.

For each analog, the FSSP database computes the structure alignment of the protein and the representative protein. This alignment establishes a 1-1 correspondence between residues in the two proteins. Taking advantage of the 1-1 correspondence for a fold family $S = \{S_1, S_2, \ldots, S_n\}$ we assign indices to residues in proteins in S as follows:

- For S_1, we assign an index i to its ith residue in the primary sequence.
- For an analog S_j ($j \in [2, n]$), we assign an index i to a residue r if r aligns with the ith residue in S_1 according to FSSP.

Residues that do not align with any residues in the representative protein do not receive any index.

10.2.5 Protein Contact Maps

Giving the index structure discussed previously, for each protein $S_j \in S$, we built a labeled graph $G = (V, E, \Sigma, \lambda)$ to represent the protein where:

- V is the set of nodes in G. Each node represents an indexed residues in S_j.

- E is the set of edges in G. Each edge represent a "contact" between a pair of residues in S_j.
- $\Sigma = \{1, 2, \ldots, m\}$ is a set of labels in G. Each label is an index and m is the length of the representative protein in S.
- $\lambda : V \rightarrow \Sigma$ maps nodes (residues) to their labels as discussed previously.

In our graph representation, two residues are in "contact" if the minimal (non-hydrogen) atom-pair distance between the residues is less than an upper-bound, e.g. 4.5 Å. This graph representation is the *residue-level contact map* of the protein S_j.

Since each node in a graph representation has an index as its label, in the following discussion, we use "site" I to refer to a node with index I in a protein, or a group nodes with the same indices I in multiple proteins. The difference is usually clear from the context. In addition, in our graph representations of proteins, there is always a one-to-one correspondence between amino acid residues of a protein and nodes in its graph representation. Because of the correspondence, we use the two terms "residue" and "node" interchangeably in the following discussion.

10.2.6 Sequence Entropy

Given a protein S_j in a fold family $S = \{S_1, S_2, \ldots, S_n\}$, we retrieve the sequences of its homologs from the HSSP database (**H**omology-derived **S**econdary **S**tructure of **P**roteins) available at http://www.cmbi.kun.nl/gv/hssp/. We compute sequence entropies according to [DS01]. In the following discussion for the purpose of self-completeness, we summarize our way of compute sequence entropy. Further details can be found in [DS01].

For each site I in $S_j \in S$, HSSP computes the probability of different amino acid residues types that appear at I. These probabilities can be conveniently written as a l-dimensional vector $V_I^j = p(a_1), p(a_2), \ldots, p(a_l)$. In this notation, a_k ($k \in [1, l]$) is an amino acid type, e.g. Ala or Leu, l is the total number of residues types, and $p(a_k)$ ($0 \leq p(a_k) \leq 1$, $\sum_{k \in [1,l]} p(a_k) = 1$) is the probability that type a_k appears at site I as computed in HSSP. The dimensionality l is 20 if we use all 20 residue types, or 6 if we use a six-letter alphabet for amino acid types. We use the six-letter alphabet used in [DS01] throughout our experiments. V_I^j is the *probability profile* of site I in structure S_j.

Given a fold family $S = \{S_1, S_2, \ldots, S_n\}$ and a site I, we compute the *mean probability profile* V_I of site I:

$$V_I = \sum_{1 \leq j \leq n} V_I^j / n \tag{10.1}$$

It is possible some structures do not contain the site I. In such cases, we adjust n to be the total number of structures used in the summation.

The *sequence entropy* $\mathcal{E}_I$at site I is the entropy of the mean probability profile $V_I = p(a_1), p(a_2), \ldots, p(a_k)$ defined as:

$$\mathcal{E}_I = -\sum_{k\in[1,l]} p(a_k) \cdot log(p(a_k))/log(l) \tag{10.2}$$

where l is the total number of residues types we use. Dividing the entropy by $log(l)$ we normalize $\mathcal{E}_I$ to $[0, 1]$.

10.2.7 Structure Entropy

Because structure tends to be far more conserved than sequence, and is preserved across analogs, estimating structure entropy would be a very helpful tool for quantitatively determining the structure conservation. However in contrast to sequence entropy, there is no obvious definition for structure entropy and the target of this section is to define a reasonable one. We achieve this goal by defining neighbor sets, projected neighbor sets, projected entropies, and structure entropies, as discussed in the following 4 subsections.

Neighbor Sets

Definition 10.2.2 *Given a site I and a structure P, any site that is in contact with I in P is its* ***neighbor*** *in P. We use symbol N_I^P denoting the neighbor set for site I in the structure P. In an n-protein fold family $S = \{S_1, S_2, \ldots, S_n\}$, the* ***neighborhood*** *N_I of site I is the union of all of its neighbor sets in S, or*

$$N_I = \bigcup_{1\le i\le n} N_I^{S_i} \tag{10.3}$$

For example in Table 10.2, we show the neighbor sets of site 1 in structures shown in Figure 10.1. The neighborhood of site 1 is $N_1 = \{2, 3, 4\} \cup \{2, 3\} \cup \{2, 4\} = \{2, 3, 4\}$.

Structure	Neighbor Set
G_1	$\{2, 3, 4\}$
G_2	$\{2, 3\}$
G_3	$\{2, 4\}$

Table 10.2: The neighbor sets for site 1 of structures in Figure 10.1.

Projected Neighbor Sets

Definition 10.2.3 *Given a structure P, a site I, and a set U, the* ***projected neighbor set*** *of site I with respect to U is $N_I^P|_U = N_I^P \cap U$. U is the* ***reference frame*** *of the projection.*

Example 10.2.1 *We show in Table 10.3 the projected neighbor sets for site 1 with different reference frames. For example, with* $U = \{2, 3\}$*, the projected neighbor set for site 1 in structure 1 is* $\{2, 3, 4\} \cap \{2, 3\} = \{2, 3\}$*, that in structure 2 is* $\{2, 3\} \cap \{2, 3\} = \{2, 3\}$*, and that in structure 3 is* $\{2, 4\} \cap \{2, 3\} = \{2\}$*.*

Structure	Projected neighbor sets		
	$U = \{2\}$	$U = \{3\}$	$U = \{2, 3\}$
G_1	$\{2\}$	$\{3\}$	$\{2, 3\}$
G_2	$\{2\}$	$\{3\}$	$\{2, 3\}$
G_3	$\{2\}$	$\emptyset$	$\{2\}$

Table 10.3: The projected neighbor sets for site 1 with different reference frames U.

Projected Entropy

Given an n-protein fold family $S = \{S_1, S_2, \ldots, S_n\}$, a site I, and a reference frame $U \subseteq N_I$, we create a random variable Y in the following way:

- Y takes value from the power set 2^U of U.
- We assign probability to a $y \in 2^U$ such that $p(y) =\mid \{i \mid y = N_I^{S_i}|_U, i \in [1, n]\} \mid /n$

The *entropy* $H(Y)$ of the random variable Y is defined as

$$H(Y) = -\sum_{y \in 2^U} p(y) \cdot log(p(y))/ \mid U \mid \tag{10.4}$$

We divide $H(Y)$ by $\mid U \mid$ to normalize it to [0, 1]. Since the random variable Y has a 1-1 correspondence with the reference frame U, we also call the entropy $H(Y)$ as the *projected entropy* of site I with respected to the reference frame U and denote it by $H(I)|_U$.

Example 10.2.2 *With the data shown in Table 10.3 and a reference frame* $U = \{2\}$*, in the following we show that the projected entropy* $H(1)|_{U=\{2\}}$ *is 0. First, we create a random variable* Y *that takes value from the set* $\{\emptyset, \{2\}\}$*. By definition* $p(\{2\})$ *is 1. This is because the projected neighbor sets of site 1 in all of the three structures are always the set* $\{2\}$*. Similarly* $p(\emptyset) = 0$*. Therefore the projected entropy* $H(1)|_{U=\{2\}}$ *= -*$(p(\{2\}) \cdots log(p(\{2\}))$ *+* $p(\emptyset) \cdot log(p(\emptyset))$ */1 = 0.*

This example shows that site 1 has "homogenous" neighbor sets if we "project" the neighbor sets to a smaller set $\{2\}$. This homogeneousness of the projected neighbor sets is measured by the projected entropy. To further illustrate the definition, in Table 10.4 we compute the projected entropy of different reference frames that are shown in Table 10.3.

Reference Frame	$U = \{2\}$	$U = \{3\}$	$U = \{2, 3\}$
$H_U(I)$	0	0.92	0.46

Table 10.4: Projected entropies of different reference frames

Structure Entropy

Definition 10.2.4 *Given a fold family S and a site I, the* ***structure entropy*** *of site I, denoted by ξ_I is the minimal projected entropy of all of its projected entropies, or*

$$\xi_I = \min_{U \neq \emptyset \subseteq N_I} H(I)|_U \tag{10.5}$$

where N_I is the neighborhood of site I.

10.2.8 Conserved Contact Subnetworks

In the previous section, we define structure entropy. In this section, we apply the recently developed frequent subgraph mining techniques [HWP03, **?**] to discovering conserved contact subnetworks in fold families. Intuitively, sites with low sequence entropy should involve in the conserved contact subnetworks and sites that participate in conserved contact subnetworks should have low structure entropy. Therefore discovering frequent contact subnetworks will offer a way to evaluate our structure entropy definition. For more details about how to identify frequent subgraphs, refer to Chapter 5.

10.3 Results

10.3.1 Construct Protein Contact Maps

For each fold family we mentioned in Section 10.2, we computed their contact maps. We tabulate the characteristics of the data sets which include the representative proteins (PDBIDs), the lengths of the representative proteins, the total number of analogs in a fold family, and the statistics about contact maps in fold families in Table 10.5.

10.3.2 Contact Probabilities

For a pair of sites (I, J) in a fold family, we estimate the contact probability $p(I, J)$ that sites I and J are in contact by dividing the total number of (I, J) contacts to the total number of proteins contains both sites I and J. Using Ig family, we illustrate the contact probabilities in Figure 10.2.

These contact probabilities reveal important information about the structure conservation in the fold family. For example, Figure 10.2 shows that there are highly conserved non-local contacts among two segments that are centered at site 30 and site 70 in the Ig family. These

Fold	Ig	OB	R	$\alpha/\beta - P$	TIM
PDBid	1TEN	1MJC	3CHY	2ACY	2EBN
L	89	69	128	98	285
N	205	76	626	129	193
V_{max}	89	69	128	98	285
V_{min}	46	36	43	46	72
V_{mean}	71	55.7	90.6	66.9	154.4
V_{σ}	9.6	7.7	16.6	8.9	45.4
E_{max}	394	283	606	437	1407
E_{min}	156	118	153	137	259
E_{mean}	270.5	206.6	361.7	263.5	625.9
E_{σ}	52.3	38.2	90.3	48.5	236.6
D_{max}	18	16	18	18	19
D_{min}	1	1	1	1	1
D_{mean}	7.6	7.4	8.0	7.9	8.1
D_{σ}	2.9	2.9	2.9	2.8	3.0

Table 10.5: Statistics collected for the contact maps in 5 fold families. L: length of a representative protein. N: total number of analogs in a fold family. V_max, V_min,V_mean, V_{σ} are the maximal, minimal, mean, and standard deviation of node number in contact maps in a fold family, respectively. The same statistics are collected for edges (E) and node degrees (D).

contacts belongs to two anti-parallel β sheet strands centered at sites 30 and 70. Similarly, contacts between the two anti-parallel β sheet strands centered at sites 15 and 55 are also high conserved in the fold family.

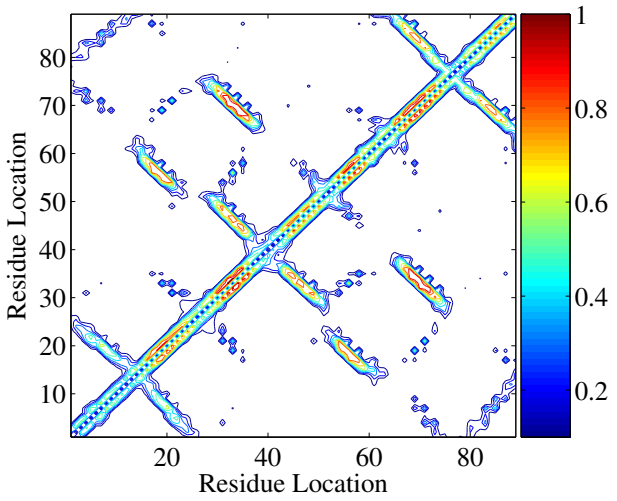

Figure 10.2: Contact probabilities of the Ig fold families.

10.3.3 Compute Structure Entropy

In the previous section, we compute contact probability that measures the stability of a single contact. In order to quantitatively measure the structure stability in protein structures in a fold family, we compute the structure entropy for each site according to Section 10.2.7. In our computation in order to remove noise and to reduce computational complexity, we remove contacts whose contact probabilities are less than $0 \leq p \leq 1$, where p is a parameter. Thus in this settings, for a site I in the fold family, we define a family of structure entropies $\xi_I(p)$ controlled by the parameter p.

In Figure 10.4 for the Ig family, we show the structure entropies computed with different values of p for all sites in the Ig family. By the thick line with "$\times$" marker, we show the structure entropies with p= 0.4. The upper-boundary line is structure entropy distribution with $p = 0.5$ and the lower-boundary line is that of $p = 0.25$. Though structure entropy values at a given site may differ slightly with different parameter values, the overall distributions are not sensitive to p. In addition, at low entropy sites, such as sites 33 and 70, the entropy values remain the same regardless of which parameter value we use. If we increase the value of the parameter p to be more than 0.5, some sites do not have any contacts with such high probability and hence the structure entropies are not well-defined (data not shown). If we reduce p to a lower value such as 0.1, the structure entropy value may be quite different as shown in Figure 10.4. With the above observations, we set $p = 0.4$ in all the subsequent studies.

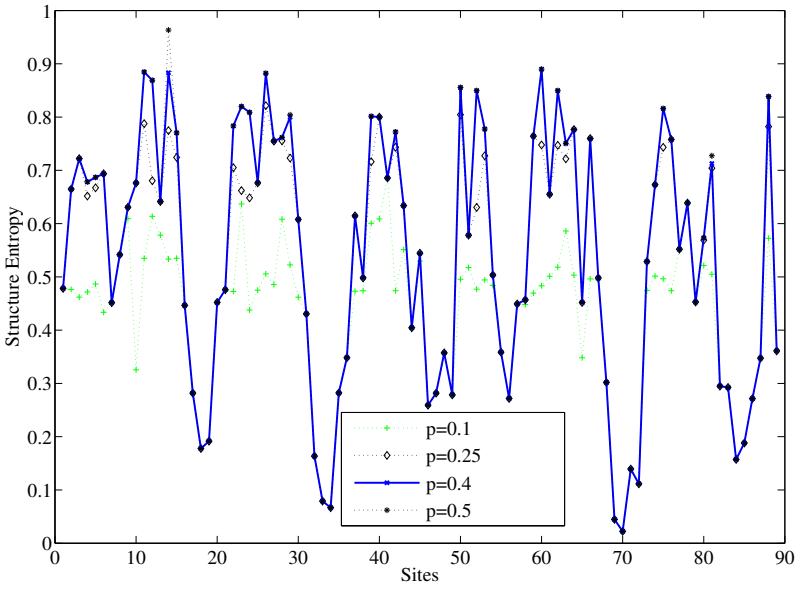

Figure 10.3: Structure entropies in the Ig fold family.

Comparing Figure 10.2 and Figure 10.4, we notice that low entropy sites in Figure 10.4 such as sites 19, 33, 57, and 70, correspond to regions with high contact probability Figure 10.2. This observation indicates that our structure entropy definition provides a reasonable measurement of the structure conservation since it correlates well with simple statistics such as contact probability. The utility of our structure entropy is further illustrated in the next section where we study the quantitative relationship between sequence and structure entropies.

In Figure 10.7, we show contact probabilities and structure entropies for fold family OB, Rossman, $\alpha/\beta - P$, and Tim Barrel, respectively.

10.3.4 Correlation Between Sequence Entropy and Structural Conservation

For each fold family, we compute the sequence entropy for the representative protein according to Section 10.2.6. For the Ig family, the sequence entropies of all sites in the family are shown in Figure 10.4. In order to compare sequence and structure entropy, in the same figure we also plot structure entropy. Matching the two entropies, we notice that lower sequence entropy always coupled with lower structure entropy. For example, site 70 has a very low sequence entropy (0.5) and also has a very low structure entropy (close to 0). However, the opposite is not always true. For example, site 71 has a very high sequence entropy (0.9) but still has a very low structure entropy since it is located in a structurally conserved region of the protein.

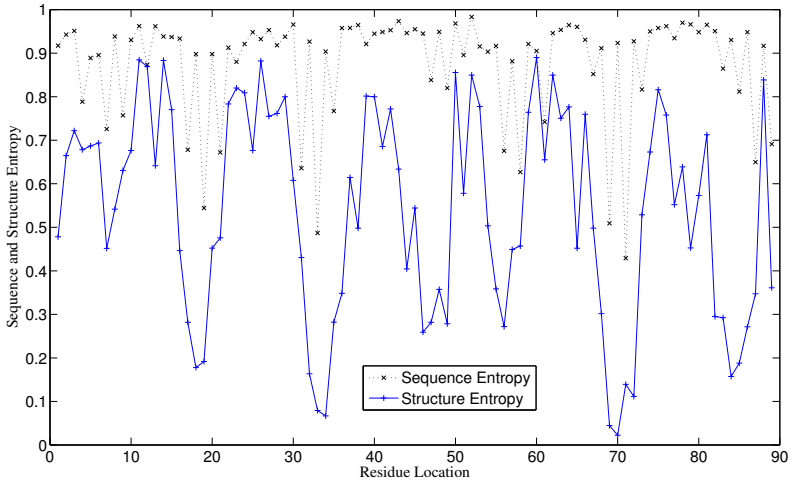

Figure 10.4: Sequence and structure entropies in the Ig fold family.

The asymmetric relation between sequence conserved regions and structure conserved regions is not hard to understand. It is well known that sequence conserved sites play important role in the structure (and/or function) of the protein. On the other hand, sites in a highly

conserved structure region may still undergoes fast mutations and that's the reason why there are so many proteins with diverse sequences that can still adopt the same fold. In the sequel, we call sites that are conserved in both sequence (characterized by low sequence entropy) and structure (characterized by low structure entropy) "influential". For example, sites 33, 68, and 70 in the Ig family are "influential".

In Figure 10.5 for the Ig family, we further study the correlation between the sequence entropy and the structure entropy using linear regression. At the left part of Figure 10.5, we show the scatter plot of the sequence entropy (x-axis) and the structure entropy (y-axis) with the fitted linear regression line. If sites with low sequence entropy always couple with low structure entropy and vice versa, we should expect a positive correlation between the two entropies. With a few outliers (to be discussed), this is the case for the Ig family. The correlation coefficient is 0.48 and under the null hypothesis that there is no correlation, the *P–value* of the observed linear regression is 1.9×10^{-6}. Since there is no obvious reason to believe the linear relationship between the sequence entropy and structure entropy, the regression line and the *P–value* computation have only illustrative purpose.

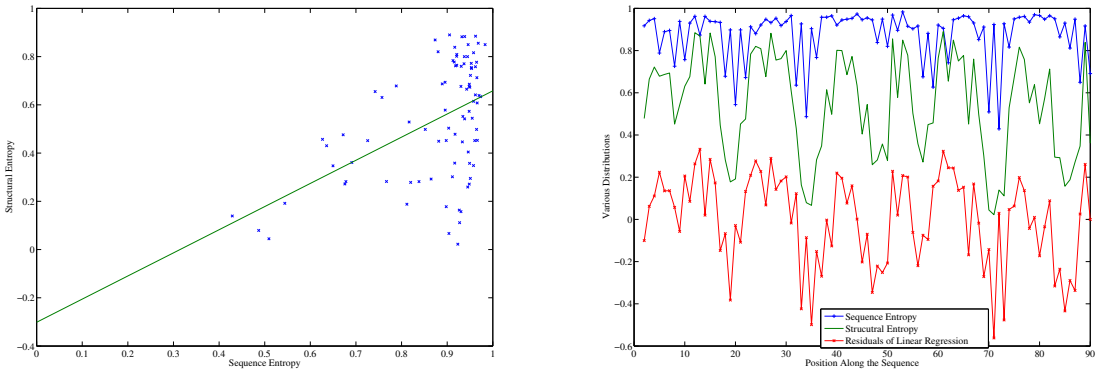

Figure 10.5: Correlation between sequence and structure entropies. Left: the scatter plot of the sequence entropy (x-axis) and the structure entropy (y-axis) with the fitted linear regression line. Right: distribution of sequence entropy (blue), structural entropy (green), and linear regression residual (red) of sites in the Ig fold family.

We postulate that the asymmetric relation between sequence entropy and structure entropy contributes to the low correlation we observe. In other words, sites that are close to an influential site tend to have low structure entropy (since they are located in a highly conserved structure region) but high sequence entropy. To confirm this, we utilize a simple method to eliminate "outliers" by removing sites which have high sequence entropy (¿0.8) and low structure entropy (¡0.4), if they are in direct contact with at least one an influential residue. This leads to an improved correlation where the correlation coefficient increases from 0.48 to 0.74 and the *P–value* of the linear regression drops from 1.9×10^{-6} to 6×10^{-14}.

We carried the similar tests for the rest of the fold families and the results are shown in Table 10.6.

Fold	Ig	OB	R	$\alpha/\beta - P$	TIM
r	0.74	0.62	0.74	0.73	0.68
p	6×10^{-14}	$1.3 \times 10{-}6$	$< 10^{-15}$	5.9×10^{-14}	$< 10^{-15}$

Table 10.6: Linear correlation of sequence and structure entropy. Both correlation and statistical significance were computed using Matlab. $< 10^{-15}$ was used in this table when zero was output by Matlab for the statistical significance.

10.3.5 Structure Entropy and Conserved Contact Subnetworks

Given a group of contact maps, the focus of this section is to use the graph mining method to locate the most invariant contact subnetwork(s) across protein structures in a fold family. To that end, we use the algorithm called FFSM (Fast Frequent Subgraph Mining) [HWP03]. The algorithm supports a group of parameters, among which two are important for our task: (1) frequency threshold σ, which indicates that a mined subgraph pattern should appears in a minimal σ fraction of the group of graphs and (2) closeness $\mathcal{D}$ to complete graphs, which indicate that at most $\mathcal{D}$ number of edges for a subgraph pattern may miss from a complete graph. A complete graph is one where all nodes are pair-wise joined and is also called *clique* in graph theory. In the context of a contact map, cliques (or ones close to them) correspond to substructures where residues are extensively interacting.

In Figure 10.6, we show one of 5 most frequent cliques we obtained from the Ig family. This clique is composed by four residues: Thr836, Try837, Glu870, and Val871. Two residues (Try837 and Val871) in the cliques are believed to be the folding nucleus of the protein [HSC00]. The other four cliques are shown in Table 7.

All the residues that covered by the 5 cliques correspond to sites 33-35 and sites 69-73 in the Ig fold family. These sites all have low structure entropy and some of them have low sequence entropy (e.g. sites 35 and 70). This offered an additional evidence for correlated sequence and structure evolution.

Cliques	Residues
1	Glu834, Leu835, SER872, Tyr873
2	Thr836, Val871, SER872, Glu870
3	Glu834, Leu835, Thr836, Ser872
4	Glu834, SER872, Tyr873, Ile874

Table 10.7: Most frequent cliques in the Ig family with frequency 162 in the total of 205 proteins

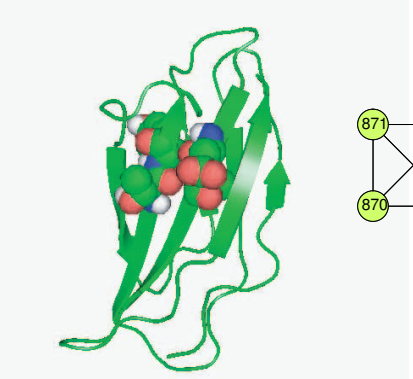

Figure 10.6: One of the pattern obtained in protein 1TEN with support 164. Left: the occurrence of the pattern in the protein 1TEN. Right: the topology of the pattern. This pattern is a fully connect graph.

10.4 Discussions

To compute the structure entropy of a site I, we search for the minimal projected entropy which is defined on every subset of the neighbor hood N_I of I. The total number of such projected entropy is therefore . For each element in the power set of N_I, we need to scan the full neighbor sets of I and to compute the entropy. Therefore the overall time complexity is where $\mid D \mid$ is the total number of items in the neighbor sets of I. In order to speed up the computation, we perform the following optimizations:

- Sort items X in NI in descending order according to the total number of occurrences of X in the neighbor sets of I.
- Organize the search space of as a tree rather than a lattice, as did in [HWP03].

In the future, we plan to investigate more efficient optimization techniques, such as branch-and-bound to reduce the search space.

10.5 Conclusions

In this chapter we study correlated sequence and structure evolution in large fold families. We developed a novel structure entropy measurement to quantitatively measure the structure conservation across structures in a fold family. As shown by the results, our structure entropy definition is stable in the presence of structure uncertainties and correlates well with sequence entropy. Applying graph mining techniques, we identify highly conserved contact groups that may contribute to the folding of the structures. In our future study, we will improve computational efficiency of both structure entropy computation and contact pattern discovery.

We will also investigate the utility of structure entropy and structure patterns in structure prediction.

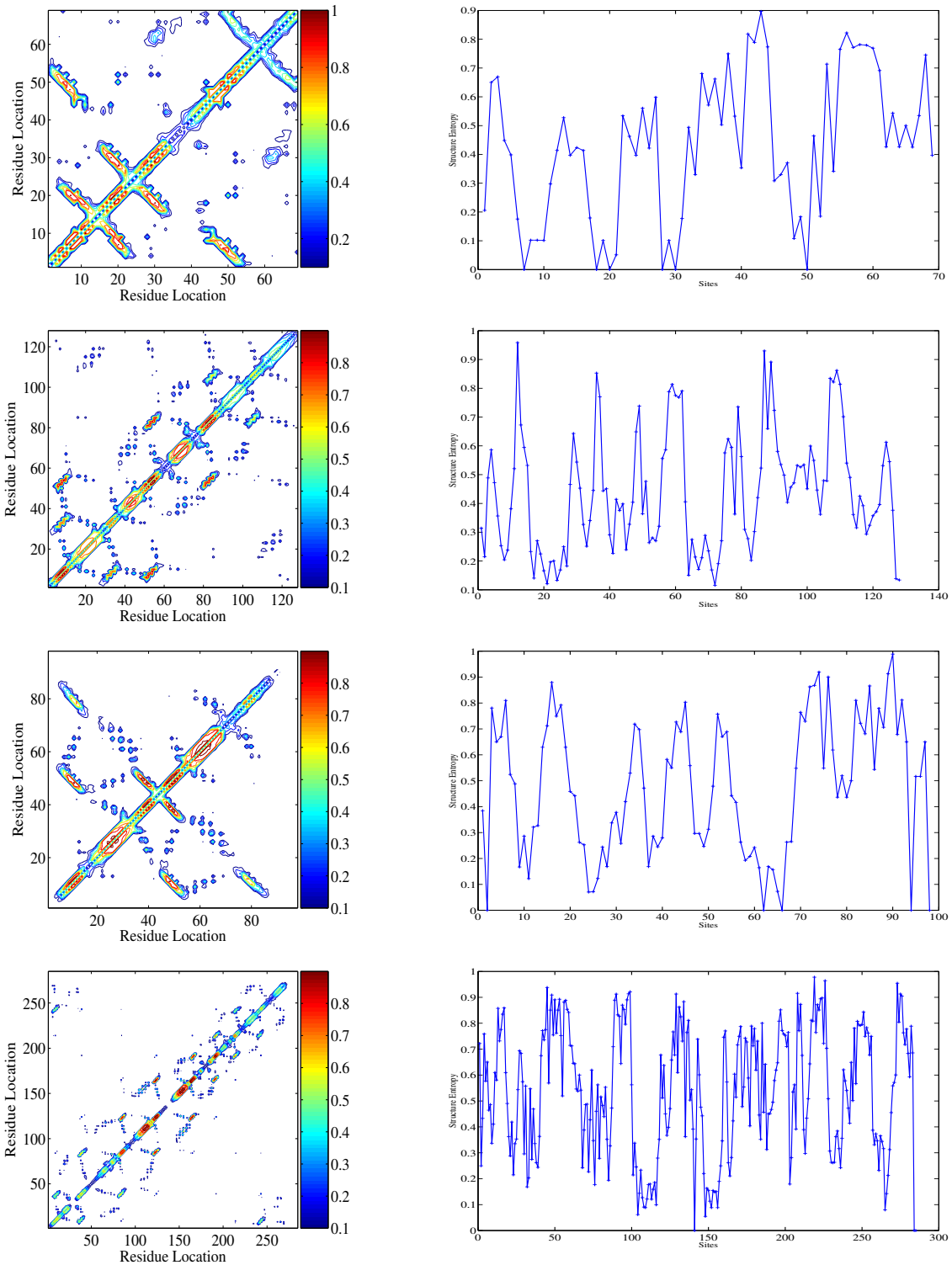

Figure 10.7: From top to bottom: various distributions for the fold families: OB, R, $\alpha/\beta - P$, and TIM Barrel. Two graphs are shown for each fold family. Left: the contact probabilities. Right: the structure entropies.

Chapter 11

Conclusions and Future Directions

11.1 Conclusions

Structure comparison of proteins has been a major bioinformatics research topic with various biological applications including structure classification, function annotation, functional site identification, protein design, and protein engineering.

In studying structure comparison, new computational techniques have been identified and some of these techniques are applicable for a general application domain outside bioinformatics.

In the future, we expect to witness the successes of structure comparison in both algorithmic improvements and new applications. Our optimistic view is based on the following two factors:

- Computers are becoming more powerful.
- the recently started proteomics research will rapidly produce a large volume of structure and structure-related data.

Below, we review the plausible future directions that we think are important for structure comparison.

11.2 Future Directions

Here we review the possible future direction of structure comparison in two subdirections: (1) identifying applications in the biological/biomedical domain, (2) developing new computational techniques.

11.2.1 Future Applications

Three future applications of structure comparison are discussed.

Dynamic Protein Structures

There is no question that understanding the dynamics of proteins structures offers great information for biological research. For example, enormous insights can be gained if we can directly observe the process of protein folding using experimental techniques [Wei00].

Currently, the Nuclear Magnetic Resonance spectroscopy (NMR) is the major experimental technique to measures a protein's native structure in a solvent environment. NMR determines the average protein structure by measuring the distances among protons and specially labeled carbon and nitrogen atoms [PR04]. NMR has been applied to obtain protein structure, protein-protein complexes, and protein-ligand complexes which account for approximately 10% of the overall structures in PDB. There are also several specialized methods that have been developed to report the dynamic structure of proteins in specialized biological processes such as protein folding and domain movement in multi-domain proteins [Wei00, HCA00].

Protein dynamics brings significant opportunities to the current structure comparison method because of the rich information stored in the trajectory of protein structures. We envision two types of comparisons: *intra structure comparison*, which analyzes the protein structure motion and detects important features for a single protein, and *inter structure comparison*, which compares dynamics data for multiple protein structures and identifies common features.

Though techniques to collect structure dynamics data are in their infancy stage, we believe that such techniques, as well as computational methods for molecular dynamics, will mature in the future rapidly and be successful in helping domain experts gain useful insights into various biological processes.

Protein-protein Interaction

Protein-protein interaction refers to the ability of proteins to form complexes. Protein-protein interaction data is usually formed as an undirected graph whose nodes are proteins and edges connect two protein if the proteins can form a stable/transient complex [AM13].

Protein-protein interaction data bring new challenges for structure comparison. In order to elucidate common structural motifs involved in protein-protein interaction and finally to predict the interaction computationally, we need to compare multiple protein complexes rather than single structures. We also need to be able to define the boundary of the interaction, based on the structure of the complexes.

Subcellular Localization

Knowledge about where a protein may be located in a cell is of paramount importance for biological research and pharmaceutical companies. For example, an outer membrane protein

is one that is transported to the outer membrane after its synthesis. Knowing a protein is an outer membrane protein simplifies the drug design process since outer membrane proteins can be accessed easily by drugs [GSW+03]. As another example, knowing the localization of a protein offers important information for assembling metabolic pathways [SFne].

Predicting the subcellular localization is one of the active research topics in bioinformatics research [GSW+03, NN94, SFne]. Protein subcellular localization has been investigated in two ways. The first approach relies on sequence motifs as descriptors to assign subcellular localization for protein sequences. This approach is based on the observation that continuous stretches of amino acid residues may encode the signal that guides a protein to a specific location. The second approach utilizes the amino acid composition of proteins to predict the possible localization. This technique is motivated by the observation that residue composition of a protein highly correlates with the localization of the proteins [NN94].

Recently there is evidence showing that protein structure is also important for predicting the related subcellular localization. For example, the β-barrel is known as a signature for outer membrane proteins. This observation has resulted in significantly improvement of the prediction accuracy, as reported in [GSW+03]. As another example, the FKBP-type peptidyl prolyl cis-trans isomerase (PPIase) is a large group of proteins with 4 possible subcellular localizations. As reported by Himukai *et al.*, the subcellular localization of these proteins is correlated with the conserved structure domain around the active sites of the protein [HKH99]. As shown in this preliminary study, incorporating structure comparison can improve the accuracy of the protein subcellular prediction.

11.2.2 New Computational Techniques

Facing the challenges of handling large and complex structure data, we believe new computational techniques will be invented for structure comparison. The possible directions are (1) developing approximate matching in pattern discovery, (2) inventing efficient index structure to speed up pattern matching in a structure database, (3) devising new data visualization techniques for structure comparison, (4) integrating data from different sources for structure comparison, and (5) statistical structure comparison. Below we discuss statistical structure comparison as the end of the dissertation.

Comparison with Statistical Analysis

As shown in sequence analysis methods, statistical models such as Hidden Markov Model (HMM) are useful for recognizing sequence similarity that is not easily detectable by straightforward alignment methods. Given the success of statistical tools in sequence comparison, it is natural to consider extending those tools (and possibly to introduce new ones) for structure comparison of proteins.

Here we review a recently developed algorithm 3dHMM [AG04] whose goal is to build a rigorous description of protein 3D structure family using HMM. In outline, 3dHMM takes a group of aligned 3D structure and a query structure as inputs and computes the best alignment of the query structure to the structure group in the following way: (1) estimating the 3D Gaussian for each position (the C_α atom in each amino acid residue) of the aligned structures, (2) estimating the deletion probability for each position using the aligned structures (assuming the alignment is not gap-free), (3) using a modified Viterbi algorithm [RJ86] to find the best alignment of the query structure to the HMM model, and (4) use the Forward algorithm [RJ86] to calculate the probability that the query structure was generated from the HMM model. The 3dHMM method has been applied to several protein families and has achieved better results in terms of identifying structure homology than the traditional RMSD calculation.

There are many other types of statistical analysis tools, such as Markov Random Field [Bes74], Hidden Markov Random Field, and Bayesian Networks [HD96]. It is interesting to check their applicability in protein structure comparison.

BIBLIOGRAPHY

[AG04] V Alexandrov and M. Gerstein. Using 3d hidden markov models that explicitly represent spatial coordinates to model and compare protein structures. *BMC Bioinformatics.*, 9(5), 2004.

[AK98] L. Aravind and E. V. Koonin. Phosphoesterase domains associated with DNA polymerases of diverse origins. *Nucleic Acids Res*, 26:3746–3752, 1998.

[AM13] Ruedi Aebersold and Matthias Mann. Mass spectrometry-based proteomics. *Nature*, 422:198–207, 2003 March 13.

[APG+94] Peter J. Artymiuk, Andrew R. Poirrette, Helen M. Grindley, David W. Rice, and Peter Willett. A graph-theoretic approach to the identification of three-dimensional patterns of amino acid side-chains in protein structures. *Journal of Molecular Biology*, 243:327–44, 94.

[AQAS01] P. Aloy, E. Querol, F. X. Aviles, and M. J. Sternberg. Automated structure-based prediction of functional sites in proteins: applications to assessing the validity of inheriting protein function from homology in genome annotation and to protein docking. *J Mol Biol*, 311:395–408, 2001.

[AS94] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *20th Int. Conf. on Very Large Databases (VLDB)*, pages 487–99, 1994.

[Bai00] A. Bairoch. The enzyme database in 2000. *Nucleic Acids Res.*, 28(1):304–305, January 2000.

[BB02] C. Borgelt and M. R. Berhold. Mining molecular fragments: Finding relevant substructures of molecules. In *Proc. International Conference on Data Mining'02*, pages 51–58, 2002.

[BCG01] Douglas Burdick, Manuel Calimlim, and Johannes Gehrke. Mafia: A maximal frequent itemset algorithm for transactional databases. *ICDE*, 2001.

[BDH96] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The Quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, 22(4):469–483, 1996.

[Bes74] J. Besag. Spatial interaction and the statistical analysis of lattice systems. *J. Royal Statist. Soc. B*, 36:192–236, 1974.

[BHL+05] D. Bandyopadhyay, J. Huan, J. Liu, J. Prins, J. Snoeyink, A. Tropsha, and W. Wang. Using fast subgraph isomorphism checking for protein functional annotation using SCOP and Gene Ontology. *UNC CS technical reprot*, 2005.

[BKB02] Philip Bradley, Peter S. Kim, and Bonnie Berger. TRILOGY: Discovery of sequence-structure patterns across diverse proteins. *Proceedings of the National Academy of Sciences*, 99(13):8500–8505, June 2002.

[BS04] D. Bandyopadhyay and J. Snoeyink. Almost-Delaunay simplices : Nearest neighbor relations for imprecise points. In *ACM-SIAM Symposium On Distributed Algorithms*, pages 403–412, 2004.

[BT91] Carl Branden and John Tooze. *Introduction to Protein Structure*. Garland Publishing, 1991.

[BT03] JA Barker and JM Thornton. An algorithm for constraint-based structural template matching: application to 3d templates with statistical analysis. *Bioinformatics*, 19(13):1644–9, 2003.

[Bur00] S. K. Burley. An overview of structural genomics. *Nat Struct Biol*, 7 Suppl:932–934, 2000.

[BWF+00] H.M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T.N. Bhat, H. Weissig, I.N. Shindyalov, and P.E. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[CBB+02] M R Chance, A R Bresnick, S K Burley, J S Jiang, C D Lima, A Sali, S C Almo, J B Bonanno, J A Buglino, S Boulton, H Chen, N Eswar, G He, R Huang, V Ilyin, L McMahan, U Pieper, S Ray, Vidal M, and L K. Wang. Structural genomics: a pipeline for providing structures for the biologist. *Protein Sci*, 11:723–38, 2002.

[CCT02] SA Cammer, CW Carter, and A. Tropsha. Identification of sequence-specific tertiary packing motifs in protein structures using delaunay tessellation. *Lecture notes in Computational Science and Engineering,*, 24:477–494, 2002.

[CLA99] K. H. Choi, R. A. Laursen, and K. N. Allen. The 2.1 angstrom structure of a cysteine protease with proline specificity from ginger rhizome, zingiber officinale. *Biochemistry*, 7, 38(36):11624–33, 1999.

[Con04] Gene Ontology Consortium. The Gene Ontology (GO) database and informatics resource. *Nucl. Acids. Res.*, 32(90001):D258–261, 2004.

[CS99] MH Cordes and RT. Sauer. Tolerance of a protein to multiple polar-to-hydrophobic surface substitutions. *Protein Sci.*, 8(2):318–25, 1999.

[CYM03] Yun Chi, Yirong Yang, and Richard Muntz. Indexing and mining free trees. *ICDM*, 2003.

[DBSS00] N. V. Dokholyan, S. V. Buldyrev, H. E. Stanley, and E. I. Shakhnovich. Identifying the protein folding nucleus using molecular dynamics. *J. Mol. Biol.*, 296:1183–88, 2000.

[Del34] B. Delaunay. Sur la sphère vide. A la memoire de Georges Voronoi. *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskih i Estestvennyh Nauk*, 7:793–800, 1934.

[DFS99] A. Deutsch, M. F. Fernandez, and D. Suciu. Storing semistructured data with STORED. *in SIGMOD*, pages 431–442, 1999.

[DHR+ar] JE Donald, IA Hubner, VM Rotemberg, EI Shakhnovich, and LA. Mirny. CoC: a database of universally conserved residues in protein folds. *Bioinformatics*, 2005, to appear.

[DLS00] P. D'haeseleer, S. Liang, and R. Somogyi. Genetic network inference: from co-expression clustering to reverse engineering. *Bioinformatics*, 16(8):707–26, 2000.

[DS01] Nikolay V. Dokholyan and Eugene I. Shakhnovich. Understanding hierarchical protein evolution from first principles. *JMB*, 312:289–307, 2001.

[DTK98] L. Dehaspe, H. Toivonen, and R. D. King. Finding frequent substructures in chemical compounds. In *4th International Conference on Knowledge Discovery and Data Mining*, pages 30–36, 1998.

[DW98] G Dodson and A Wlodawer. Catalytic triads and their relatives. *Trends Biochem Sci.*, 23(9):347–352, September 1998.

[EJT04] I. Eidhammer, I. Jonassen, and W. R. Taylor. *Protein Bioinformatics: An Algorithmic Approach to Sequence and Structure Analysis*. John Wiley & Sons, Ltd, 2004.

[Fer99] Alan Fersht. *Structure and Mechanism in Protein Science*. W.H. Freeman Comp., 1999.

[FWLN94] D. Fischer, H. Wolfson, S. L. Lin, and R. Nussinov. Three-dimensional, sequence order-independent structural comparison of a serine protease against the crystallographic database reveals active site similarities: potential implication to evolution and to protein folding. *Protein Science*, 3:769–778, 1994.

[GARW93] H.M. Grindley, P.J. Artymiuk, D.W. Rice, and P. Willet. Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm. *J. Mol. biol.*, 229:707–721, 1993.

[GAW97] E. J. Gardiner, P. J. Artymiuk, and P. Willett. Clique-detection algorithms for matching three-dimensional molecular structures. *J Mol Graph Model*, 15:245–253, 1997.

[GB01] John A. Gerlt and Patricia C. Babbitt. Divergent evolution of enzymatic function: Mechanistically diverse superfamilies and functionally distinct suprafamilies. *Annu. Rev. Biochem.*, 70:20946, 2001.

[GC02] J. Gough and C. Chothia. SUPERFAMILY: HMMs representing all proteins of known structure. SCOP sequence searches, alignments and genome assignments. *Nucleic Acids Res*, 30:268–272, 2002.

[GSB+05] R.A. George, R.G. Spriggs, G.J. Bartlett, A. Gutteridge, M.W. MacArthur, C.T. Porter, B. Al-Lazikani, J.M. Thornton, and M.B. Swindells. Effective function annotation through residue conservation. *PNAS*, 102:12299–12304, 2005.

[GST+04] Richard A. George, Ruth V. Spriggs, Janet M. Thornton, Bissan Al-Lazikani, and Mark B. Swindells. Scopec: a database of protein catalytic domains. *Bioinformatics*, Supp 1:I130–I136, 2004.

[GSW+03] JL Gardy, C Spencer, K Wang, M Ester, GE Tusnady, I Simon, S Hua, K deFays, C Lambert, K Nakai, and FS Brinkman. Psort-b: Improving protein subcellular localization prediction for gram-negative bacteria. *Nucleic Acids Res.*, 31(13):3613–7, 2003.

[GW97] Roy Goldman and Jennifer Widom. Dataguides: Enabling query formulation and optimization in semistructured databases. *In VLDB'97*, 1997.

[HBW+05] J. Huan, D. Bandyopadhyay, W. Wang, J. Snoeyink, J. Prins, and A. Tropsha. Comparing graph representations of protein structure for mining family-specific residue-based packing motifs. *Journal of Computational Biology*, 12(6):657–71, 2005.

[HCA00] WL Hubbell, DS Cafiso, and C. Altenbach. Identifying conformational changes with site-directed spin labeling. *Nat Struct Biol.*, 7(9):735–9, 2000.

[HCD94] L. B. Holder, D. J. Cook, and S. Djoko. Substructures discovery in the subdue system. *Proc. AAAI'94 Workshop Knowledge Discovery in Databases*, pages 169–180, 1994.

[HD96] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *Intl. J. Approximate Reasoning*, 15(3):225–263, 1996.

[HDS96] William Humphrey, Andrew Dalke, and Klaus Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[HG99] H. Hegyi and M. Gerstein. The relationship between protein structure and function: a comprehensive survey with application to the yeast genome. *J Mol Biol*, 288:147–164, 1999.

[HK01] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and techniques*. Morgan Kaufmann Publishers, 2001.

[HKH99] R Himukai, T Kuzuhara, and M. Horikoshi. Relationship between the subcellular localization and structures of catalytic domains of fkbp-type ppiases. *J Biochem (Tokyo)*, 126(5):879–88, 1999.

[HMPL+04] H. Hermjakob, L. Montecchi-Palazzi, C. Lewington, S. Mudali, S. Kerrien, S. Orchard, M. Vingron, B. Roechert, P. Roepstorff, A. Valencia, H. Margalit, J. Armstrong, A. Bairoch, G. Cesareni, D. Sherman, and R. Apweiler. Intact - an open source molecular interaction database. *Nucl. Acids. Res.*, 32:D452–D455, 2004.

[Hor87] B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A: Optics, Image Science, and Vision*, 4(4):629–642, 1987.

[HPW+06] Jun Huan, Jan Prins, Wei Wang, Charlie Carter, and Nikolay V. Dokholyan. Coordinated evolution of protein sequences and structures with structure entropy. In *Computer Science Department Technical Report*, 2006.

[HS96] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–602., 1996.

[HSC00] SJ Hamill, A. Steward, and J. Clarke. The folding of an immunoglobulin-like greek key protein is defined by a common-core nucleus and regions constrained by topology. *J Mol Biol.*, 297(1):165–78, 2000.

[HSS+02] J. Hu, X. Shen, Y. Shao, C. Bystroff, and M. J. Zaki. Mining protein contact maps. *2nd BIOKDD Workshop on Data Mining in Bioinformatics*, 2002.

[HWB+04] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining protein family specific residue packing patterns from protein structure graphs. In *Proceedings of the 8th Annual International Conference on Research in Computational Molecular Biology (RECOMB)*, pages 308–315, 2004.

[HWP03] J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, pages 549–552, 2003.

[HWPY04] J. Huan, W. Wang, J. Prins, and J. Yang. SPIN: Mining maximal frequent subgraphs from graph databases. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 581–586, 2004.

[HWW+04] J. Huan, W. Wang, A. Washington, J. Prins, R. Shah, and A. Tropsha. Accurate classification of protein structural families based on coherent subgraph analysis. In *Proceedings of the Pacific Symposium on Biocomputing (PSB)*, pages 411–422, 2004.

[IWM00] A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructures from graph data. *In PKDD'00*, pages 13–23, 2000.

[JECT02] I. Jonassen, I. Eidhammer, D. Conklin, and W. R. Taylor. Structure motif discovery and mining the PDB. *Bioinformatics*, 18:362–367, 2002.

[JET99] I. Jonassen, I. Eidhammer, and W. R. Taylor. Discovery of local packing motifs in protein structures. *Proteins*, 34:206–219, 1999.

[Jon97] I. Jonassen. Efficient discovery of conserved patterns using a pattern graph. *Comput Appl Biosci.*, 13(5):509–22, 1997.

[JT04] Susan Jones and Janet M Thornton. Searching for functional sites in protein structures. *Current Opinion in Chemical Biology*, 8:3–7, 2004.

[Kab78] W.A. Kabsch. Discussion of solution for best rotation of two vectors. *Acta Crystallography A.*, 34:827–828, 1978.

[KBD+58] JC Kendrew, G Bodo, HM Dintzis, RG Parrish, H Wyckoff, and DC. Phillips. A three-dimensional model of the myoglobin molecule obtained by x-ray analysis. *Nature*, 181:662–666, 1958.

[KK01] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proc. International Conference on Data Mining'01*, pages 313–320, 2001.

[KMS00] LA Kelley, RM MacCallum, and MJ Sternberg. Enhanced genome annotation using structural profiles in the program 3d-pssm. *J Mol Biol.*, 299(2):499–520, 2000.

[KMWO05] J. Ko, L. F. Murga, Y. Wei, and M. J. Ondrechen. Prediction of active sites for protein structures from computed chemical properties. *Bioinformatics*, 21 Suppl 1:i258–i265, 2005.

[KQHG03] L. N. Kinch, Y. Qi, T. J. Hubbard, and N. V. Grishin. CASP5 target classification. *Proteins*, 53 Suppl 6:340–351, 2003.

[KT03] Bala Krishnamoorthy and Alexander Tropsha. Development of a four-body statistical pseudo-potential to discriminate native from non-native protein conformations. *Bioinformatics*, 19(12):1540–48, 2003.

[KWK04] E. V. Koonin, Y. I. Wolf, and G. P. Karev, editors. *Power Laws, Scale-free Networks and Genome Biology*. Springer, 2004.

[LFNW01] N. Leibowitz, ZY Fligelman, R. Nussinov, and HJ Wolfson. Automated multiple structure alignment and detection of a common substructural motif. *Proteins*, 43(3):235–45, May 2001.

[LPR01] Andrei N. Lupasa, Chris P. Pontingb, and Robert B. Russellc. On the evolution of protein folds: Are similar motifs in different protein folds the result of convergence, insertion, or relics of an ancient peptide world? *Journal of Structural Biology*, 134:191–203, 2001.

[LWT05a] R. A. Laskowski, J. D. Watson, and J. M. Thornton. ProFunc: a server for predicting protein function from 3D structure. *Nucleic Acids Res*, 33:W89–93, 2005.

[LWT05b] Roman A A. Laskowski, James D D. Watson, and Janet M M. Thornton. Protein function prediction using local 3d templates. *J Mol Biol*, 351:614–626, July 2005.

[Mat96] BW Matthews. Structural and genetic analysis of the folding and function of t4 lysozyme. *The FASEB Journal*, 10:35–41, 1996.

[MBHC95] A.G. Murzin, S.E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–40, 1995.

[MPP04] EC Meng, BJ Polacco, and Babbitt PC. Superfamily active site templates. *Proteins*, 55(4):962–76, 2004.

[MSO03] M Milik, S Szalma, and KA. Olszewski. Common structural cliques: a tool for protein structure and function analysis. *Protein Eng.*, 16(8):543–52., 2003.

[MVK+04] M. Madera, C. Vogel, S. K. Kummerfeld, C. Chothia, and J. Gough. The SUPERFAMILY database in 2004: additions and improvements. *Nucleic Acids Res*, 32:D235–9, 2004.

[NK04] S. Nijssen and J.N. Kok. A quickstart in frequent structure mining can make a difference. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 647–652, 2004.

[NKGP90] David J. Neidhart, George L. Kenyon, John A. Gerlt, and Gregory A. Petsko. Mandelate racemase and muconate lactonizing enzyme are mechanistically distinct and structurally homologous. *Nature*, 347:692–4, 1990.

[NN94] H Nakashima and K. Nishikawa. Discrimination of intracellular and extracellular proteins using amino acid composition and residue-pair frequencies. *J Mol Biol.*, 238(1):54–61, 1994.

[NOT02] N Nagano, CA Orengo, and JM Thornton. One fold with many functions: the evolutionary relationships between tim barrel families based on their sequences, structures and functions. *Journal of Molecular Biology*, 321:741–765, 2002.

[NW91] Ruth Nussinov and Haim J. Wolfson. efficient detection of three-dimensional structural motifs in biological macromolecules by computer vision techniques. *PNAS*, 88:10495–99, 1991.

[OJT03] C. Orgengo, D. Jones, and J. Thornton. *Bioinformatics: genes, proteins, and computers.* BIOS Scientific Publishers ltd., 2003.

[OMD+99] R. Overbeek, Fonstein M., M. D'Souza, G.D. Pusch, and N. Maltsev. The use of gene clusters to infer functional coupling. *Proc Natl Acad Sci*, 96(6):2896–901, 1999.

[OMJ+97] C.A. Orengo, A.D. Michie, S. Jones, D.T. Jones, M.B. Swindells, and J.M. Thornton. CATH - a hierarchic classification of protein domain structures. *Structure*, 5(8):1093–1108, 1997.

[PA98] X. Pennec and N. Ayache. A geometric algorithm to find small but highly similar 3d substructures in proteins. *Bioinformatics*, 14(6):516–22, 1998.

[PBO+05] S. C. Pegg, S. Brown, S. Ojha, C. C. Huang, T. E. Ferrin, and P. C. Babbitt. Representing structure-function relationships in mechanistically diverse enzyme superfamilies. *Pac Symp Biocomput*, pages 358–369, 2005.

[PBZ+13] Eric Phizicky, Philippe I. H. Bastiaens, Heng Zhu, Michael Snyder, and Stanley Fields. Protein analysis on a proteomic scal. *Nature*, 422:208–15, 2003 March 13.

[PDZH02] Jian Pei, Guozhu Dong, Wei Zou, and Jiawei Han. On computing condensed frequent pattern bases. *ICDM*, 2002.

[PE05] D. Pal and D. Eisenberg. Inference of protein function from protein structure. *Structure (Camb)*, 13:121–130, 2005.

[PR04] Gregory A Petsko and Dagmar Ringe. *Protein Structure and Function*. New Science Press Ltd, Middlesec House, 34-42 Cleveland Street, London W1P 6LB, UK, 2004.

[PS04] F. Pazos and M. J. Sternberg. Automated prediction of protein function and detection of functional sites from structure. *Proc Natl Acad Sci U S A*, 101:14754–14759, October 2004.

[RGM03] S. Raghavan and H. Garcia-Molina. Representing web graphs. *In Proceedings of the IEEE Intl. Conference on Data Engineering*, 2003.

[Ric81] J S. Richardson. Class-directed structure determination: foundation for a protein structure initiative. *Adv Protein Chem*, 34:167–339, 1981.

[RJ86] L. R. Rabiner and B. H. Juang. An introduction to hidden Markov models. *IEEE ASSP Magazine*, pages 4–15, January 1986.

[RSKC05] J Roach, S Sharma, M Kapustina, and CW Jr. Carter. Structure alignment via delaunay tetrahedralization. *Proteins*, 60(1):66–81, July 2005.

[Rus98] Robert B. Russell. Detection of protein three-dimensional side-chain patterns: new examples of convergent evolution. *Journal of Molecular Biology*, 279:1211–1227, 1998.

[SAW03] R. V. Spriggs, P. J. Artymiuk, and P. Willett. Searching for patterns of amino acids in 3D protein structures. *J Chem Inf Comput Sci*, 43:412–421, 2003.

[SBV01] K. Shearer, H. Bunks, and S. Venkatesh. Video indexing and similarity retrieval by largest common subgraph detection using decision trees. *Pattern Recognition*, 34(5):1075–1091, 2001.

[SDDS03] BE Shakhnovich, NV Dokholyan, C DeLisi, and EI. Shakhnovich. Functional fingerprints of folds: evidence for correlated structure-function evolution. *J Mol Biol.*, 326(1):1–9, 2003.

[SFne] G Schneider and U. Fechner. Advances in the prediction of protein targeting signals. *Proteomics*, 4(6):1571–80, 2004 June.

[SFK00] Jeffrey Skolnick, Jacquelyn S. Fetrow, and Andrzej Kolinski. Structural genomics and its importance for gene function analysis. *nature biotechnology*, 18:283–287, 2000.

[SGR04] M. H. Serres, S. Goswami, and M. Riley. GenProtEC: an updated and improved analysis of functions of Escherichia coli K-12 proteins. *Nucleic Acids Res*, 32:D300–2, 2004.

[Sha95] J. P Shaffer. Multiple hypothesis testing. *Ann. Rev. Psych*, pages 561–584, 1995.

[SIK+04] Roded Sharan, Trey Ideker, Brian P. Kelley, Ron Shamir, and Richard M. Karp. Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data. *ACM RECOMB*, pages 282–289, 2004.

[SKBW98] JM Schwehm, ES Kristyanne, CC Biggers, and Stites WE. Stability effects of increasing the hydrophobicity of solvent-exposed side chains in staphylococcal nuclease. *Biochemistry*, 37(19):6939–48, 1998.

[SKK02] S. Schmitt, D. Kuhn, and G. Klebe. A new method to detect related function among proteins independent of sequence and fold homology. *J. Mol. Biol.*, 323(2):387–406, 2002.

[SKMS97] A. Srinivasan, R. D. King, S. H. Muggleton, and M. Sternberg. The predictive toxicology evaluation challenge. *In Proc. of the 15th International Joint Conferrence on Artificial Intelligence (IJCAI)*, pages 1–6, 1997.

[SMM+11] Chakkaravarthi S, Babu MM, Gromiha MM, Jayaraman G, and Sethumadhavan R. Exploring the environmental preference of weak interactions in (alpha/beta)(8) barrel proteins. *Proteins.*, 2006 Jul 11.

[SPNW04] A Shulman-Peleg, R Nussinov, and HJ Wolfson. Recognition of functional sites in protein structures. *Journal of Molecular Biology*, 339(3):607–633, jun 2004.

[SR03] A Stark and RB Russell. Annotation in three dimensions. pints: Patterns in non-homologous tertiary structures. *Nucleic Acids Res*, 31(13):3341–4, 2003.

[SS91] C Sander and R. Schneider. Database of homology-derived protein structures and the structural meaning of sequence alignment. *Proteins*, 9(1):56–68, 1991.

[SSK+05] R Sharan, S Suthram, RM Kelley, T Kuhn, S McCuine, P Uetz, T Sittler, RM Karp, and T. Ideker. Conserved patterns of protein interaction in multiple species. *Proc Natl Acad Sci*, 102(6):1974–9, 2005.

[SSPNW05] Maxim Shatsky, Alexandra Shulman-Peleg, Ruth Nussinov, and HJ Wolfson. Recoginition of binding patterns common to a set of protein structures. *to appear in RECOMB*, 2005.

[SSR98] Alexander Stark, Shamil Sunyaev, and Robert B. Russell. A model for statistical significance of local similarities in structure. *Journal of Molecular Biology*, 326:1307–1316, 1998.

[SSR04] A. Stark, A. Shkumatov, and R. B. Russell. Finding functional sites in structural genomics proteins. *Structure (Camb)*, 12:1405–1412, 2004.

[STV96] R.K. Singh, A. Tropsha, and I.I. Vaisman. Delaunay tessellation of proteins. *J. Comput. Biol.*, 3:213–222, 1996.

[TBPT05] James W. Torrance, Gail J. Bartlett, Craig T. Porter, and Janet M. Thornton. Using a library of structural templates to recognise catalytic sites and explore their evolution in homologous families. *J. Mol. Biol.*, 347:565581, 2005.

[TCCV03] A. Tropsha, C.W. Carter, S. Cammer, and I.I. Vaisman. Simplicial neighborhood analysis of protein packing (SNAPP) : a computational geometry approach to studying proteins. *Methods Enzymol.*, 374:509–544, 2003.

[TJ04] William R. Taylor and Inge Jonassen. A method for evaluating structural models using structural patterns. *Proteins*, July 2004.

[TOK+03] A. Teplyakov, G. Obmolova, P. P. Khil, A. J. Howard, R. D. Camerini-Otero, and G. L. Gilliland. Crystal structure of the Escherichia coli YcdX protein reveals a trinuclear zinc active site. *Proteins*, 51:315–318, 2003.

[TWP+98] T C Terwilliger, G Waldo, T S Peat, J M Newman, K Chu, and J. Berendzen. Class-directed structure determination: foundation for a protein structure initiative. *Protein Sci*, 7:1851–1856, 1998.

[Ull76] J. R. Ullman. An algorithm for subgraph isomorphism. *Journal of the Association for Computing Machinery*, 23:31–42, 1976.

[VFB+00] A Via, F Ferre, B Brannetti, A Valencia, and M. Helmer-Citterich. Three-dimensional view of the surface motif associated with the p-loop structure: cis and trans cases of convergent evolution. *Journal of Molecular Biology*, 303(4):455–465, November 2000.

[VGS02] N. Vanetik, E. Gudes, and E. Shimony. Computing frequent graph patterns from semi-structured data. *Proc. International Conference on Data Mining'02*, 2002.

[vMHJ+03] C von Mering, M Huynen, D Jaeggi, S Schmidt, P Bork, and B. Snel. String: a database of predicted functional associations between proteins. *Nucleic Acids Res.*, 31:258261, 2003.

[WBL+03] Z. Wang, G. Benoit, J. Liu, S. Prasad, P. Aarnisalo, X. Liu, H. Xu, NP. Walker, and T. Perlmann. Structure and function of nurr1 identifies a class of ligand-independent nuclear receptors. *Nature*, 423(3):555–60, 2003.

[WBT97] AC Wallace, N Borkakoti, and JM Thornton. Tess: a geometric hashing algorithm for deriving 3d coordinate templates for searching structural databases. application to enzyme active sites. *Protein Sci*, 6(11):2308–23, 1997.

[WD03] G. Wang and R. L. Dunbrack. PISCES: a protein sequence culling server. *Bioinformatics*, 19:1589-1591, 2003. http://www.fccc.edu/research/labs/dunbrack/pisces/culledpdb.html.

[Wei00] S. Weiss. Measuring conformational dynamics of biomolecules by single molecule fluorescence spectroscopy. *Nature Structural Biology*, 7(9):724–9, 2000.

[WKHK04] N. Weskamp, D. Kuhn, E. Hullermeier, and G. Klebe. Efficient similarity search in protein structure databases by k-clique hashing. *Bioinformatics*, 20:1522–1526, 2004.

[WSO01] M Weir, M Swindells, and J. Overington. Insights into protein function through large-scale computational analysis of sequence and structure. *Trends Biotechnol*, 19(10 suppl):s61–6, 2001.

[WTR+03] PP Wangikar, AV Tendulkar, S Ramya, DN Mali, and S Sarawagi. Functional sites in protein families uncovered via an objective and automated graph theoretic approach. *J Mol Biol*, 326(3):955–78, 2003.

[WY05] Wei Wang and Jiong Yang. Mining sequential patterns from large data sets. *Series of Advances in Database Systems*, 2005.

[YH02] X. Yan and J. Han. gspan: Graph-based substructure pattern mining. In *Proc. International Conference on Data Mining'02*, pages 721–724, 2002.

[YH03] X. Yan and J. Han. Closegraph: Mining closed frequent graph patterns. *KDD'03*, 2003.

[Zak02a] M. J. Zaki. Efficiently mining frequent trees in a forest. *In SIGKDD'02*, 2002.

[Zak02b] Mohammed Zaki. Efficiently mining freqeunt trees in a forest. *SIGKDD*, 2002.