Local Structure Comparison of Proteins

JUN HUAN, JAN PRINS, AND WEI WANG

Department of Computer Science
University of North Carolina at Chapel Hill
USA
huan@cs.unc.edu
prins@cs.unc.edu
weiwang@cs.unc.edu

Abstract

Protein local structure comparison aims to recognize structural similarities between parts of proteins. It is an active topic in bioinformatics research, integrating computer science concepts in computational geometry and graph theory with empirical observations and physical principles from biochemistry. It has important biological applications, including protein function prediction. In this chapter, we provide an introduction to the protein local structure comparison problem including challenges and applications. Current approaches to the problem are reviewed. Particular consideration is given to the discovery of local structure common to a group of related proteins. We present a new algorithm for this problem that uses a graph-based representation of protein structure and finds recurring subgraphs among a group of protein graphs.

ISSN: 0065-2458/DOI: 10.1016/S0065-2458(06)68005-4

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 [21,62,68,76]. 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 [35].

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 *local structure comparison* to identify structural similarity between parts of proteins [23].

This chapter provides an introduction to the protein structure comparison problem, focusing on recent research on local structure comparison. Work in this area combines computational geometry and graph theory from computer science with empirical observations and physical principles from biochemistry. The protein structure comparison problem has important applications in classification and function

prediction of proteins, and is also of use in protein folding research and rational drug design [49].

The chapter is organized as follows. In the remainder of this section we describe the factors driving the need for protein structure comparison and present the structure comparison problem, and our area of focus. Section 2 outlines the necessary biological background, including a high-level introduction to protein sequence, structure, and function. Readers with limited knowledge of proteins and protein structure may wish to read this section before proceeding further. In Section 3 we present a taxonomy of current algorithms for the problem of protein local structure comparison. In Section 4, we give an introduction to graph representations of protein structure, and describe how discovering common local structure may be viewed as a data mining problem to identify frequent subgraphs among a collection of graphs. In Section 5, we introduce an efficient subgraph mining algorithm. Results obtained using graph-based local structure comparison on various key problems in protein structure are presented in Section 6. Finally we conclude in Section 7 with some thoughts on future directions for work in this area. This chapter also includes an extensive bibliography on protein structure comparison.

1.1 Motivation

This section describes the factors that underscore the need for automated protein structure comparison methods.

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 [13,94]. The result has been a rapid increase in the number of known 3D protein structures. The Protein Data Bank (PDB) [6], 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 5000 structures were deposited 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 [52].

Along with individual protein structures, the structure of certain complexes of interacting proteins are known as well. While the structures of 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 [83]. The IntAct database records 50,559 binary interactions

involving 30,497 proteins [32] 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 [100].

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 [1,73]. The rapidly growing body of data call for automatic computational tools rather than manual processing.

1.1.2 Structure Comparison Aids Experiment Design

Protein structure comparison is part of a bioinformatics research paradigm that performs comparative analysis of biological data [84]. 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 structure comparison including structure classification, functional site identification, and structure-based functional annotation. A comprehensive review can be found in [49].

1.1.2.1 Structure Classification. Classification of protein structures is vital to providing easy access to the large body of protein structures, for studying the evolution of protein structures, and for facilitating structure prediction. For example, through global structure classification, domain experts have identified many sequences that have low pairwise sequence identity yet have adopted very similar 3D structures. Such information helps significantly in structure prediction [51].

Traditionally, protein structure classification is a time consuming manual task, for example as used to construct the Structure Classification of Protein (SCOP) database [62]. SCOP is maintained using visual examination of protein structures by domain experts. With the development of automated global structure comparison methods such as CATH [68] and DALI [35], structure classification has become more automated.

In DALI and CATH, the units of classification are protein domains. Domains are organized hierarchically based on their similarity at the sequence, structure, and function level. Classification systems such as DALI and CATH utilize three common steps to derive a hierarchical grouping of protein structures. The first step is to select from all known structures a subset of "representative" structures among which (pairwise) sequence similarity is low. The second step is to compare the set of structures to compute an all-by-all similarity matrix. Based on this matrix, the third step is to perform a hierarchical clustering to group similar structures together. How to compute

the similarity between a pair of structures and how to perform hierarchical clustering are the two key components in protein classification. For example, in DALI, proteins are classified at 4 levels according to *class*, *fold*, *functional families*, and *sequence family* and in CATH, proteins are classified into 5 levels according to *class*, *architecture*, *topology*, *homology superfamilies*, and *sequence families*. Though different methods may lead to different classifications, careful comparison of classification systems has revealed that existing systems (DALI, CATH, and SCOP) overlap significantly [21].

1.1.2.2 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 [3,19,60,99]. See [95] 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 structure have been developed [95]. 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.

1.1.2.3 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 [26], 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 [105], the expression level [18], and at the level of the gene's chromosome location [70].

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 [95] as an example of using local structure comparison for function annotation.

Torrance et al. first constructed a database of functional sites in enzymes [95]. 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 [95]. With the database of functional sites, Torrance et al. then used the JESS method [5] 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.

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 structure comparison.

1.2 Challenges

We decompose the challenges associated with structure comparison into three categories: (1) the nature of protein structure data and structure representation methods, (2) the tasks in structure comparison, and (3) the computational components of structure comparison methods.

1.2.1 The Nature of Protein Structure

In order to compare 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 [21], which is a popular view in designing structure comparison 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.

1.2.1.1 Structure Representations.

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

- *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 accessible 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.
- *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.

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 [21] for further details.

1.2.2 Tasks in Structure Comparison

To outline the challenges associated with structure comparison, it is convenient to group current structure comparison methods into common tasks, according to the final goal of the comparison. The categorization we use is not unique, further division is possible, and we expect that new tasks will emerge to augment the list in the future. However, our current categorization summarizes well all the methods that we will describe in this chapter and is useful as a starting point for the introduction of structure comparison algorithms.

- *Global structure comparison*
  - Computing the alignment of a group of two or more structures.
  - Computing the overall similarity between two structures.
  - Searching a set of proteins to find those that are similar to a given protein structure.
- *Local structure comparison*
  - Identifying common substructures among a group of proteins.
  - Searching a set of proteins for occurrences of a particular substructure.
  - Searching a database of substructures for the substructures that appear in a particular protein structure.

The tasks within a specific type of structure comparison (global or local) are closely related. For example, the computation of the pair-wise global structure similarity is usually done after aligning the two structures. Tasks in different types of structure comparison can also be related. For example, in computing the global alignment of two structures, one way is to first compute the shared substructures as "seeds" and then to select and connect such set of seeds to produce the global alignment [35].

1.2.3 Components of Structure Comparison Tasks

The tasks listed in the previous section can be decomposed into a number of components. These include a basic notion of similarity between structures, or 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 of a task must be displayed in a meaningful fashion. In this section, we elaborate each of these concepts.

1.2.3.1 Defining Pattern or Structure Similarity.

A *structure pattern* is a geometric arrangement of protein elements, for example four specific amino acids placed at the vertices of a tetrahedron of specified dimensions. We list three considerations in defining similarity between structures or between a pattern and a structure.

- *Level of Structure Representation*
  We may choose atoms, amino acid residues, or secondary structure elements (SSE), as the elements for protein structure comparison. The choice of elements are made according to the specific goal of the comparison and the preference of the investigators. The general concern in choosing a detailed representation where elements are atoms or amino acid residues is that the coordinates of such

elements in protein structures are subject to experimental noise and hence any comparison algorithms should have a certain level of robustness to perturbation of the geometry of the structure. In addition, a detailed representation often leads to a more extensive computation than a coarse representation such as SSE. On the other hand, by choosing SSEs as structure elements, we may miss valuable information about a protein structure. Early structure comparison used SSE as elements extensively, mainly for the purpose of efficient computation. Recent research tends to use amino acid residues or atoms because of the detailed representation.

- *Sequence Order in Structure Comparison*
  In sequence-order dependent structure comparison, the primary sequence order of the structure elements must be preserved in a pattern or an alignment. Otherwise, we carry out a sequence-independent structure comparison.
- *Pair-Wise or Multi-Way Structure Comparison*
  In pair-wise comparison, we find the similarity of a pair of structures, or find a pattern in common to two structures. A generalization of pair-wise structure comparison is a multi-way comparison that involves more than two structures.

As a few examples, most structure alignment algorithms, such as DALI [35], compute the pairwise alignment of two structures that preserves the sequence order of structure elements and hence are sequence dependent, pair-wise global structure comparison methods. In contrast to structure alignment, most of the structure pattern discovery methods, such as those based on graphs [39], 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.

1.2.3.2 Scoring Functions. A scoring function quantifies the fitness of a structure pattern or an alignment to the observed data. Choosing the right scoring function involves a certain level of art. Ideally, the right scoring function should correlate precisely with the desired consequence of the analysis, e.g. the evolutionary connection of a pair of structures in an global alignment. Practically, such ideal scoring functions are very difficult to obtain due to the limited knowledge we have. Therefore, investigators often resort to "generic" scoring functions. For example, the root-mean-squared-deviation (RMSD) [21] is usually used to compute the closeness of two structures with a known 1–1 correspondence of structure elements in the two protein structures. In computing RMSD, we superimpose one structure onto the other such that the sum of the squared distances between corresponding elements is minimized. A closed-form definition of this scoring function can be found in [50, 36].

1.2.3.3 Search Procedures. In protein structure comparison with a given scoring function, a search procedure is often utilized to identify the best solution. One of the most widely used search procedures is the subgraph matching algorithm that determines whether a pattern (specified by one graph) matches a structure (specified by another graph) (see Section 5 for further details). Computational efficiency is the major concern for designing a search procedure.

1.2.3.4 Results Presentation. Usually the final step of structure comparison 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 structure comparison.

1.3 Our Focus in Structure Comparison

We focus on protein local structure comparison and present an overview of the frontier of the research, balancing algorithmic developments and biological applications. We single out local structure comparison because it has become popular in recent structure comparison research. The transition from global structure comparison to local structure comparison is well supported by a wide range of experimental evidence.

- *Protein function is usually carried out by a small region of the protein*. 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 [15,81]. 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 [58].

  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 [14,103, 20,53].

- *Similar global structure may not correlate with similar function*. For example, it is well known that the TIM barrels are a large group of proteins with a remarkably similar fold, yet widely varying catalytic function [63]. A striking result was reported in [65] 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 [28].

  It has also been 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 [31]. 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 [27].
- *Local similarity detection can offer evidence for 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 [77]. Convergent evolution has been studied in the serine protease family, porphyrin binding proteins [77], and the ATP/GTP binding proteins [99]. 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 [57]. Though the exact evolutionary mechanism is still debated, studying local structure similarity can help in understanding how protein structure and function evolve.

Various other interesting topics such as structure database search and structure-based functional inference are beyond the scope of this chapter and have been omitted. Topics in local structure comparison that are not covered in this chapter may be found in related books such as [21].

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 [10] for topics that are not covered here.

2.1 Protein Structure

2.1.1 Proteins are Chains of 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*) [10]. 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 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 Protein Structure is Described in Four Levels

The levels are as follows:

- *Primary structure* describes the amino acid sequence of a protein.

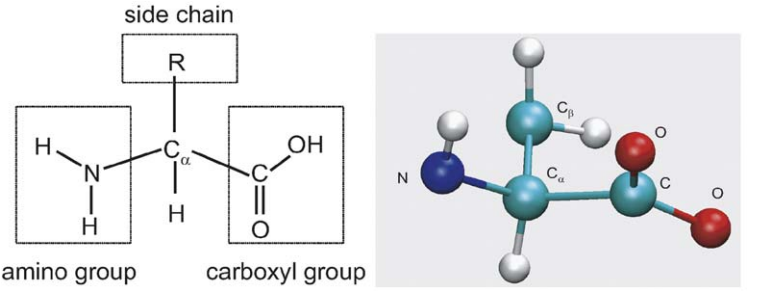

FIG. 1. Left: A schematic illustration of an amino acid. Right: The 3D structure of an amino acid (Alanine) whose side chain contains a single carbon atom. The atom types are shown; unlabeled atoms are hydrogens. The schematic diagram is adopted from [10] and the 3D structure is drawn with the VMD software.

- *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.

2.1.2.1 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 an *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 Fig. 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.

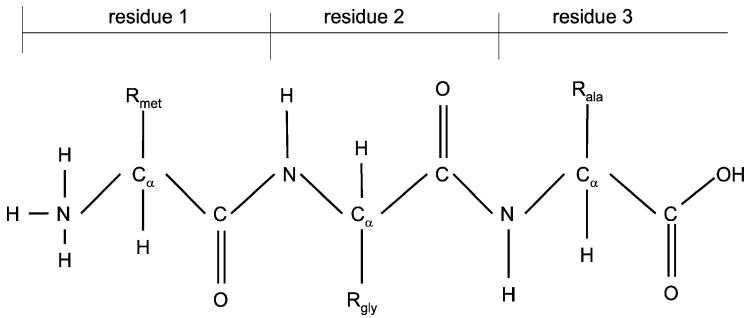

FIG. 2. A schematic illustration of a polypeptide with three residues: Met, Gly and Ala. The peptide can also be described as the sequence of the three residues: Met-Gly-Ala.

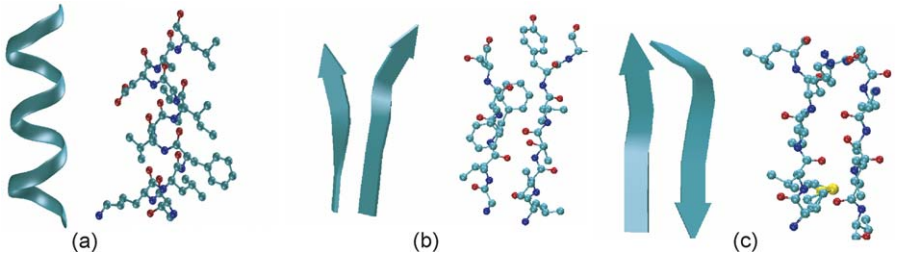

FIG. 3. A schematic illustration of the α-helix and the β-sheet secondary structures. (a) The ribbon representation of the α-helix secondary structure (on the left) and the ball-stick representation showing all atoms and their chemical bonds in the structure (on the right). We also show the same representations 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 [22]. 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.

2.1.2.2 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 Fig. 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 Fig. 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.

2.1.2.3 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.

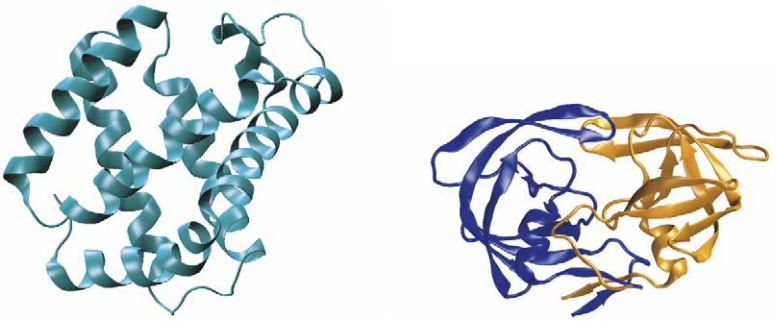

FIG. 4. Left: The schematic representation (cartoon) of the 3D structure of protein myoglobin. Right: The schematic representation (cartoon) of the 3D structure of protein HIV protease. HIV protease has two chains.

In Fig. 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).

Two types of experimental techniques are used to determine the 3D structure of a protein. In X-ray crystallography, a protein is first crystallized 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 [72]. Once the interatom distances are determined, a group of 3D structures (an *ensemble*) is computed in order to best fit the distance constraints.

2.1.3 Protein Structures are Grouped Hierarchically

2.1.3.1 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 [10]. 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 [72].

2.1.3.2 Structure Classification. 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 [72].

Here we describe a structure hierarchy, the SCOP database (Structure Classification of Proteins) [62]. SCOP is maintained manually by domain experts and considered one of the gold standards for protein structure classification. For other classification systems see [68].

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), protein 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 and 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 Fig. 5, we show a visualization developed by the Berkeley Structural Genomics Center, 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 [62].

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 [69]:

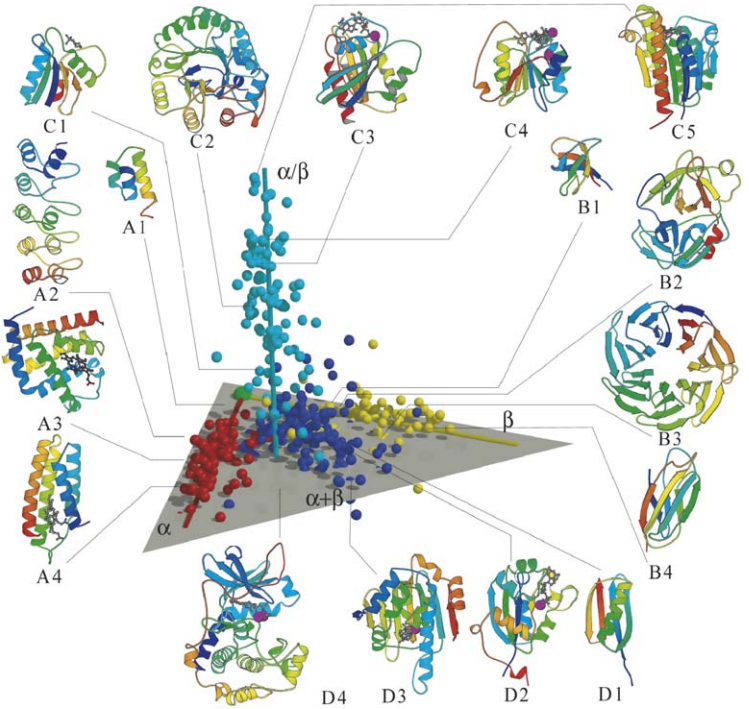

FIG. 5. The top level structural classification of proteins based on their secondary structure components. Source: http://www.nigms.nih.gov/psi/image_gallery/structures.html. Used with permission.

- *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 [72].
- *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 [26], (2) determining the molecular function is the first step in the determination of the cellular and phenotypic function of a protein.

3. A Taxonomy of Local Structure Comparison Algorithms

The goal 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.

As discussed in Section 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* [95], and *structure motifs* [21]. A typical pattern matching algorithm contains the following components:

- a definition of structure patterns;
- a scoring function that determines the fitness of a pattern to a structure;
- a search procedure that recognizes patterns in a protein or a group of proteins, based on pattern definition and the scoring function.

The scoring function is also called a *matching condition* [21]. An *instance* of a structure pattern S in a protein P is a group of amino acid residues in P that *matches* with S under a certain matching condition.

Before we proceed to details of individual algorithms, Fig. 6 presents a taxonomy of protein local structure comparison algorithms, together with sample algorithms in each category. Our categorization is not unique but it serves two purposes: (1) it offers an overview of the algorithms that are discussed in this chapter and (2) it simplifies the presentation since we find that algorithms in the same category often involve the same set of design issues.

At the top level of our taxonomy, we distinguish between pattern matching and pattern discovery algorithms. Our discussion of pattern discovery is further divided into two parts based on whether the primary sequence order of amino acid residues is significant in the pattern or not. The first group is termed *sequence-dependent* pattern discovery and the second is *sequence-independent* pattern discovery. For the more

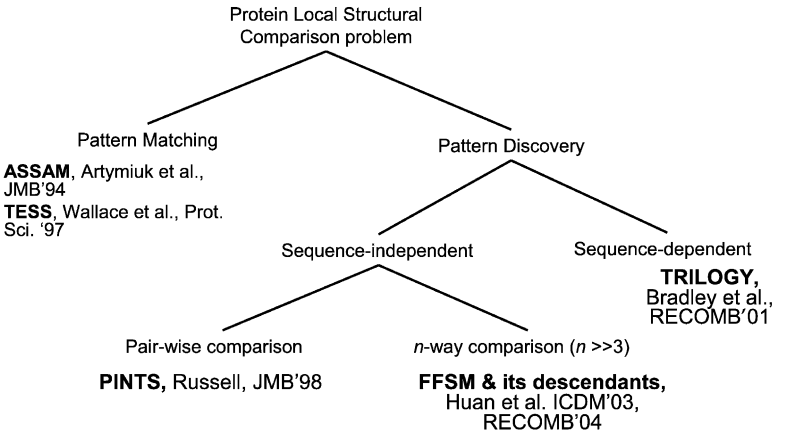

FIG. 6. A taxonomy of local structure comparison algorithms.

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 following sections survey algorithms in each category of the taxonomy.

3.1 Pattern Matching

There are three types of subproblems in pattern matching [21]:

- *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 more efficiently. In the following discussion, we present two algorithms for the complete pattern matching problem: one based on subgraph isomorphism and the other one based on geometric hashing. For probabilistic pattern matching, see [2].

3.1.1 ASSAM

The algorithm ASSAM is one of the most successful pattern matching algorithms in local structure comparison of proteins [3]. 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 graph construction and subgraph matching.

3.1.1.1 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.
- Two nodes are connected by an edge labeled by the 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 $E(R)$.

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

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

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.

3.1.1.2 Graph Matching.

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

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

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

where d_{ss}, d_{se}, d_{es}, d_{ee} are bounds on the allowed variation in distances. These inequalities help make the matching robust in the presence of experimental errors in the determination of element coordinates.

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 for which the edge labels in the pattern match the corresponding edge labels in V.

ASSAM adapts Ullman's backtracking algorithm for subgraph isomorphism [97] to solve the pattern matching problem. We discuss the details of Ullman's algorithm in Section 4.3.

3.1.2 TESS

In TESS both protein structures and structure patterns are represented as point sets, and the elements of the set are individual atoms. TESS determines whether a pattern matches a structure using geometric hashing [101]. 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 pattern structure matches the protein structure.

With minor modifications, TESS can be extended to compare a structure pattern with a group of structures. See [71] for other pattern matching algorithms that also use geometric hashing.

3.1.2.1 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 point in a 3D space and id_i is the identity of the atom.

3.1.2.2 Preprocessing in TESS.

To build a hash table encoding the geometry of a protein structure, TESS selects three atoms with their coordinates from each amino acid residue and builds a 3D Cartesian coordinate system for each selection. A 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 [101].

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 $\overrightarrow{p_1 p_2}$,
- the vector $\overrightarrow{p_1 p_2}$ 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 mapped to a value using 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 enters every atom in the protein structure into the hash table relative to this reference frame. 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.

A structure pattern in TESS is treated like a protein structure; TESS performs the same preprocessing step for a structure pattern as for a protein.

3.1.2.3 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 [101].

3.2 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 [9] and SPratt [48,47] 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]

3.2.1 TRILOGY

TRILOGY identifies sequence-dependent structure patterns in a group of protein structures [9]. There are two phases in TRILOGY: initial pattern discovery and pat-

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

tern growth. Before we discuss the two phases in details, we present the pattern definition and matching condition used in TRILOGY.

3.2.1.1 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 point representing the C_α atom in R, v is the vector of $C_\alpha C_\beta$ atoms, and id is the identity of the residue.

3.2.1.2 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.

3.2.1.3 Triplet Discovery. TRILOGY takes as inputs a group of protein structures and produces a sequence alignment of the structures using information provided in the HSSP database [78].

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 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.

3.2.1.4 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.

3.2.2 SPratt

Like TRILOGY, the SPratt algorithm also uses the primary sequence order information to detect common structure patterns in a group of protein structures [48,47]. 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.

3.2.2.1 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 point representing 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.

3.2.2.2 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.

3.2.2.3 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 [46] 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.

3.3 Sequence-Independent Pattern Discovery

3.3.1 Discovering Sequence-Independent Structure Patterns in a Pair of Structures

In the previous section, we discussed algorithms that identify sequence-dependent structure patterns. In this section, 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 [67,23].
- Graph matching methods that model protein structures as labeled graphs and perform subgraph matching to detect conserved patterns [30,61,92,89,104].

3.3.2 Geometric Hashing

This class of methods model a protein structure as point sets and use the geometric hashing technique to obtain common point subset 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 3.1.2. Below, we present the pattern definition used in geometric hashing. 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.

3.3.2.1 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 point in a 3D space that represents the spatial location of the residue (e.g. its C_α atom) and id_i is the identity of the residue.

This definition was originally used by Nussinov and Wolfson [67]. 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 and 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 $O(n^3)$.

A more efficient algorithm with complexity $O(n^2)$ has been proposed by Fischer et al. [23]. For a protein structure with n residues, rather than building a total of $O(n^3)$ (or $O(n^2)$, if using residue pairs) 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 [23].

3.3.3 Graph-Based Methods

This group of methods utilizes graph theory to model protein structure and uses subgraph isomorphism to detect recurring patterns among a pair of protein structures [91,61,79]. In this group of algorithms, a protein structure is modeled by a graph where each node models an amino acid residue, labeled by the residue identity and an edge connects a pair of residues, labeled by a variety of information related to

the geometry of the protein as well as the possible physico-chemical interactions between the pair of residues. Below we review PINTS [77,93] in detail. For related methods, see [24,61,79,107].

3.3.3.1 PINTS. PINTS takes as input two protein structures and identifies all structure patterns common to the two structures [91].

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 $(d_\alpha^{R_1,R_2}, d_\beta^{R_1,R_2}, d_f^{R_1,R_2})$ 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, C_β, and functional atoms 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

$$\begin{aligned}
\left|d_\alpha^{R_1,R_2} - d_\alpha^{R_1',R_2'}\right| &\leqslant d_\alpha, \\
\left|d_\beta^{R_1,R_2} - d_\beta^{R_1',R_2'}\right| &\leqslant d_\beta, \\
\left|d_f^{R_1,R_2} - d_f^{R_1',R_2'}\right| &\leqslant d_f
\end{aligned}$$

where d_α, d_β, d_f are predefined tolerances. PINTS uses values 7.5, 6.6, 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 [93], 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.

3.3.4 Discovering Sequence-Independent Structure Patterns in Multiple Structures

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

- Delaunay tessellation;
- Geometric hashing;
- Frequent subgraph mining.

3.3.4.1 Delaunay Tessellation.

This class of methods [54,12,96] identifies local structural 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 [4]. Recurring structural patterns can be identified from tetrahedra recurring in multiple structures. Studies have explored the hypothesis that four-residue packing motifs can be defined as structure and sequence specific residue signatures and can be utilized in annotation of structural and functional classes of both protein structures (if available) and genomic sequences [96]. Earlier studies identified residue packing patterns based on the analysis of protein structures in a family represented as a network of residue contacts obtained by Delaunay tessellation [12,42].

3.3.4.2 Geometric Hashing.

Recently geometric hashing has been applied to perform multiple structure alignment [56] and to identify functional sites in protein structures [87,85]. It has been also applied to atom-level representations of protein structures [85].

The extension of geometric hashing methods to find common structural patterns among multiple structures [87,85] and similarly for an extension based on PINTS [104] suffer from limited scalability since they may have exponential running time in the total number of structures.

3.3.4.3 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 [108,40]. We discuss frequent subgraph mining algorithms in detail in the next two sections. These methods have excellent scaling behavior as the number of structures increases.

4. Pattern Discovery Using Graph Mining

Graphs have been utilized in many application domains as a rigorous representation of real data. Such data 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. For example, the 2D structure of a chemical can be modeled as an undirected labeled graph where each node corresponds to an atom in the chemical, labeled by the atom type, and an edge corresponds to a chemical bond, labeled by the bond type. With graph representations, automated classifiers have been built to identify the toxic chemicals among a mix of toxic and non toxic chemicals [8].

Graphs have also been widely utilized for representing protein structure in protein structure comparison [3]. 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 our recent research, which combines existing graph representations of protein structures, is discussed in Section 6.

4.1 Labeled Graphs

4.1.1 Labeled Simple Graphs

We define first labeled simple graphs and then labeled multigraphs and pseudographs.

Definition 4.1. A *labeled simple graph* (graph) is a four-element tuple $G = (V, E, \Sigma, \lambda)$ 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, otherwise, G is *directed*. Unless stated otherwise, all graphs are undirected in our discussion.

4.1.2 Multigraphs and Pseudographs

A *multigraph* is a graph where there may exist at least two edges between the same pair of nodes. A *graph loop* is a degenerate edge which joins a node to itself. A simple graph can have neither loops nor multiple edges, but a *pseudograph* can have both. We define a labeled multigraph and pseudograph in the following way.

Definition 4.2. A *labeled multigraph* is a four-element tuple $G = (V, E, \Sigma, \lambda)$ where $\lambda : V \cup E \rightarrow 2^{\Sigma}$ 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 graph loops, it is a *labeled pseudograph*.

Example 1. In Fig. 7, 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 a multigraph. Graphs Q and S are simple graphs. 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.

4.1.3 Paths, Cycles, and Trees

We also use the following graph-related terms:

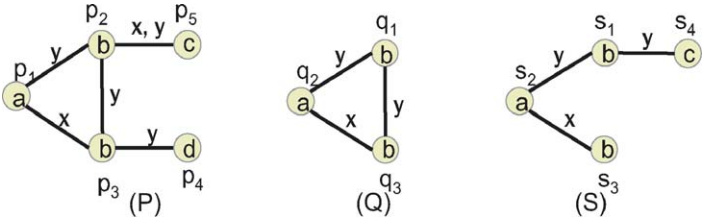

FIG. 7. A database $\mathcal{G}$ of three labeled graphs. The labels of nodes and edges are specified within the nodes and along the edges.

- 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.
- A *acyclic* graph is a graph with no cycle.
- A *tree* is a connected acyclic graph.

4.2 Representing Protein Structures

Graphs have been widely used to represent protein structures. In general at the amino acid residue level, a node in a graph represents an amino acid residue, and an edge represent the binary relation between a pair of residues. Depending on the applications, the binary relation may be distances between pairs of amino acid residues (distance matrix) or the physico-chemical contacts between residues (contact maps). We discuss the details of distance matrices and contact maps in protein structure representation below.

4.2.1 Protein Distance Matrix

A matrix $(x_{i,j})$ $(1 \leqslant i, j \leqslant n)$ 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 but the reserve is not true. Given a distance matrix X, there are at most two structures corresponding to the matrix. This is because inter-element distances are the same for a mirror image of a structure. To be efficiently handled by computer algorithms, distances in a distance matrix are discretized.

Using a distance matrix at the residue level, a protein structure is represented by a graph where a node represents an amino acid residue and an edge connecting a pair of amino acid residue is labeled by the discretized distance between the two residues.

4.2.2 Protein Contact Maps

A protein contact map is the same as the protein distance matrix representation, except each $x_{i,j}$ is not a distance but rather 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 certain distance threshold and not otherwise [37]. More sophisticated methods such as Delaunay Tessellation and almost-Delaunay are also used to define the contact relation [42].

4.3 Subgraph Isomorphism

A fundamental part of recurring subgraph identification is to decide whether a pattern G occurs in a graph G'. To make this more precise, we use the follow definition.

Definition 4.3. A graph G is *subgraph isomorphic* to another graph 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 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 $f : V[G] \rightarrow V \subseteq V[G']$ such that $E = (V \times V) \cap E[G']$. We call a graph G an *induced subgraph* of G' if G is induced subgraph isomorphic to G'.

Example 2. In Fig. 8, we show three graphs that are duplicated from Fig. 7 for the readers' convenience. The function $f : q_1 \rightarrow p_2, q_2 \rightarrow p_1$, and $q_3 \rightarrow p_3$ is

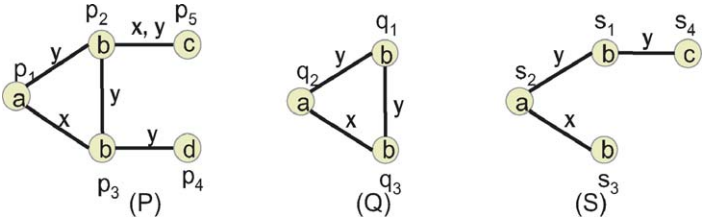

FIG. 8. A database $\mathcal{G}$ of three labeled graphs duplicated from Fig. 7. The label(s) of nodes/edges are specified within the nodes/along the edges.

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 [97]. Though Ullman originally developed the algorithm for unlabeled and undirected graphs, this algorithm is so flexible that it can be used for virtually all 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 [38] 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 graph A and B respectively and $a_{i,j}$ equals 1 if the ith node and the jth node of A are connected and 0 otherwise. Throughout this section, we use $a_{i,j}$ to refer to the entry of a matrix A at the ith row and the jth column.

Ullman used a specially designed $n \times m$ binary matrix M, referred to as the *permutation matrix*, where each row has exactly one 1 and each column has at most a single 1, to encode a 1–1 mapping from nodes of A to those of B. To see that M stands for a 1–1 mapping, 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 to 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:

$$\forall(i, j\colon\ 1 \leqslant i, j \leqslant n,\ a_{ij} = 1\ \Rightarrow\ c_{ij} = 1). \tag{1}$$

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 [97].

4.4 A Road Map of Frequent Subgraph Mining

Because graphs are ubiquitous data types in many application domains including protein structure analysis [40,39], identifying recurring patterns of graphs has attracted much recent research interest. Recurring subgraph patterns provide insights of the underlying relationships of the objects that are modeled by graphs 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 [17], efficient indexing [29,86], and web information management [110,75]. With no surprise, algorithms for graph based modeling and analysis are going through a rapid development [39].

Here, we introduce an efficient algorithm for mining graph databases: Fast Frequent Subgraph Mining (FFSM) [40]. With minor modifications, this same algorithm can be used to mine trees, cliques, quasi-cliques from a graph database or tree patterns in a tree database [40]. Before we introduce the details of our algorithm, we define the frequent subgraph mining problem, followed by an introduction to related work.

4.4.1 The Frequent Subgraph Mining 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 $s(G)$, is the fraction of $\mathcal{G}$ in which G occurs.

The frequent subgraph mining problem is defined as:

Definition 4.4. Given a graph database $\mathcal{G}$ and a parameter $0 < \sigma \leqslant 1$, 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 referred to as a *frequent subgraph mining algorithm*. 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 [55].

4.4.2 Overview of Existing Algorithms

Since frequent subgraph mining is computationally challenging, early research focused on either approximation techniques such as SUBDUE [34] or methods that are only applicable for small databases like Inductive Logic Programming [16].

Recent research in frequent subgraph mining focuses on the efficiency of the algorithms because most of the algorithms solve exactly the same problem and produce

the same answer. All scalable algorithms take advantage of the *anti-monotonicity* of frequency, which asserts that any supergraph of an infrequent subgraph pattern remains infrequent. The algorithms contain 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. frequent single node, single edge, or 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 left 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 enumerate graphs 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 smaller subset of frequent subgraphs. For example, maximal frequent subgraph mining [41] identifies only those frequent subgraphs for which none of their supergraphs are frequent. Coherent subgraph mining uses mutual information to select subgraphs that may be infrequent in an overall data set [42]. For a more recent review of different subgraph mining algorithms, see [41].

4.4.3 Edge Based Frequent Subgraph Mining

4.4.3.1 Level-wise Search: The FSG Algorithm. FSG (Frequent Subgraph Mining) [55] 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 a subsequent step, e.g. at step k, FSG identifies the set of frequent subgraphs with edge size (i.e. number of edges) k. This set is denoted as C_k. The task at step k is subdivided into two phases: candidate subgraph processing and candidate subgraph validation, with the details covered below (see Algorithm 1).

1: $F_1 \leftarrow \{e \mid s(e) \geqslant \sigma\}$ # *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 1. FSG$(\mathcal{G}, \sigma)$: Frequent subgraph mining.

Candidate Subgraph Proposing. Given a set of frequent graphs with edge size $k-1$ (number of edges), denoted by F_{k-1}, FSG constructs candidate frequent subgraphs with edge size k by "joining" two frequent subgraphs with size $k-1$. 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:

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

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 Fig. 9 and the pseudo code is presented in Algorithm 2.

Candidate Subgraph Validation. FSG determines the true frequent subgraphs with edge size k from the set C_k by computing the support value of each member

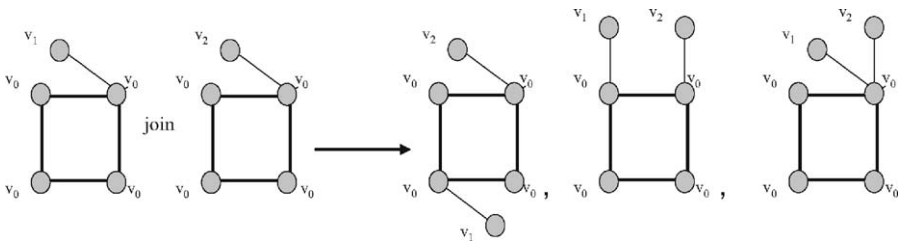

FIG. 9. An example of the join operation in FSG.

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

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

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** $s(G) \geqslant \sigma$ **then** $F_k \leftarrow F_k \cup \{G\}$ **end**
8: **end for**
9: **return** F_k

ALGORITHM 3. FSG-validation($C_k, \mathcal{G}, \sigma$): Validate frequent subgraphs.

in the set 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 of G' and updates the support value of G if it is. As the results of the validation phase, the set of frequent subgraph with edge size k is computed. The pseudo code of the FSG-validation is presented in Algorithm 3.

Putting It All Together. Algorithms 1–3 present the pseudo code for the FSG algorithm, which identifies all subgraphs F in a graph database $\mathcal{G}$ with support threshold $0 < \sigma \leqslant 1$. We simplified the FSG algorithm to explain its basic structure; see [55] for details of performance improvements in FSG.

4.4.3.2 Depth-First Search: The gSpan Algorithm.

gSpan utilizes a depth-first algorithm to search for frequent subgraphs [108]. 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 subgraphs. In order to understand the extension operation developed by gSpan, we will introduce 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 chain of nodes in G and we denote the nodes in $V[G]$ as $1, 2, \ldots, n$ where n is the size of the graph G. Node n is the *rightmost* node and the path from root to n is named the rightmost path.

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 $j_1 \leqslant j_2$).

Given a graph G and a depth-first search S, we may sort edges in a graph 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 the graph G. There may be many depth first codes for a graph G and the smallest one (using lexicographical order of sequences) is the *canonical DFS form* 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 choosing a node v in the rightmost path of a canonical DFS tree in G and adding an edge (v, w) to G where w is a node in G or not. The restriction that we only introduce an edge into the rightmost path looks strange at the first glance but an important observation of gSpan is that it is guaranteed that we can still enumerate all frequent subgraphs with this extension. See [108] for the detailed proof.

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, which leads to an order of magnitude speedup in several benchmarks [109].

Putting It All Together. Algorithms 4–6 present the gSpan algorithm.

Other Edge-Based Depth-First Algorithms. Instead of enumerating all the subgraph isomorphisms, the method proposed by Borgelt and Berhold [8] also uses an edge-based depth-first scheme to discover all frequent subgraphs. Different from gSpan, the method keeps a list of all subgraph isomorphisms ("embedding") of a frequent subgraph G. The intuition is to avoid subgraph isomorphism testing, which generally becomes the performance limiting factor of gSpan when dealing with large and complex graphs (dense graphs with few distinct labels). Another edge-based depth first search method FFSM [40] also keeps embedding and frequent subgraph.

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

ALGORITHM 4. gSpan($\mathcal{G}, \sigma$): Frequent subgraph mining.

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

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

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

ALGORITHM 6. gSpan-extension(G, k).

FFSM has developed a hybrid candidate proposing algorithm with both a join and an extension operation with improved efficiency. We cover details of FFSM in Section 5.

4.4.3.3 *Path-Based Frequent Subgraph Mining.*

Below we introduce the algorithm proposed by Vanetik et al. that discovers all frequent subgraphs using paths as a building block [98]. 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.

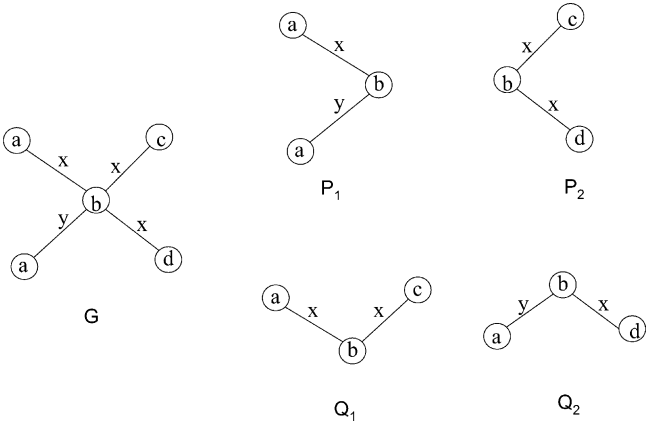

FIG. 10. A graph G and two of its path covers.

The computation of a path number is straightforward. For a connected graph $G = (V, E)$, the path number is $p(G) = |\{v \mid v \in E, d(v) \text{ is odd }\}|/2$ where $d(v)$ is the degree of a node v [98].

In Fig. 10, 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.

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 three-element tuple $G = (V, P, \pi)$ where

- V is the set of nodes in G,
- P is a path cover of G, and
- $\pi : \mathcal{P} \to V$ 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 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 [98].

Candidate Subgraph Proposing. 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 $|G_1 \cap G_2| \geqslant n - 1$.

For two joinable graphs G_1, G_2, PGM produces a set of graphs that are supergraphs to both G_1 and G_2 and selects those that are frequent in a graph database. PGM follows the general approach of Algorithm 1, using this definition of joining.

4.4.3.4 Tree-Based Frequent Subgraph Mining: the GASTON Algorithm.

We describe the algorithm GASTON [66], which introduced a new frequent subgraph enumeration method by first identifying all frequent trees and then constructing 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 with two edges, infrequent trees are pruned, and the same search goes on 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 require $i < j$ 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 and Kok, the task of constructing frequent cyclic graphs from a tree T is similar to the frequent item set mining problem [11] 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.

5. FFSM: Fast Frequent Subgraph Mining

Here, we introduce an efficient algorithm for mining frequent subgraphs in graph databases: Fast Frequent Subgraph Mining (FFSM). With little effort, this same algorithm can be used to mine trees, cliques, quasi-cliques from a graph database or tree patterns in a tree database [40].

5.1 New Definitions

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 graph normalization, i.e. assigning unique ids for graphs. To that end, we introduce the following definitions.

Definition 5.1. 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 maps 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 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 is no edge. In the sequel with no confusion of graphs, we use capital letters to denote matrices and use the corresponding lower case letters with subscripts to denote an individual entry of a matrix. For instance, we use $m_{i,j}$ to denote the entry on the ith row and jth column of an $n \times n$ matrix M, where $0 < j \leqslant i \leqslant n$.

5.1.2.1 Code. In general there are many valid adjacency matrix for a single graph. For example, any permutation of the node set corresponds to a (possibly different) adjacency matrix, if we layout the nodes along the diagonal line of the adjacency matrix accordingly. Therefore, there may be up to $n!$ different adjacency matrices for a graph of n nodes. The right part of Fig. 11 shows 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 define a total order of all adjacency matrices for a graph.

Definition 5.2. 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 s formed by concatenating lower triangular entries of M (including entries on the diagonal) where $s = m_{i,j}$ where $1 \leqslant j \leqslant i \leqslant n$.

For an adjacency matrix M, each diagonal entry of M is referred to as a *node entry* and each off-diagonal none-zero entry in the lower triangular part of M is referred

to as an *edge entry*. We order edge entries according to their relative positions in the code of the matrix M in such way that 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 3. In Fig. 11, 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.

5.1.2.2 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* (CAM) of graph G', denoted by $\mathcal{M}(G)$. For example, after applying the total ordering, we have $code(M_1) =$ "$axbxyb0yyb$" $\geqslant code(M_2) =$ "$axb0ybxyyb$" $\geqslant code(M_3) =$ "$bybyyb0xxa$." Therefore the adjacency matrix M_1 shown in Fig. 11 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 [55,45] in the above canonical form definition. This definition provides important properties for subgraph mining, as explained below.

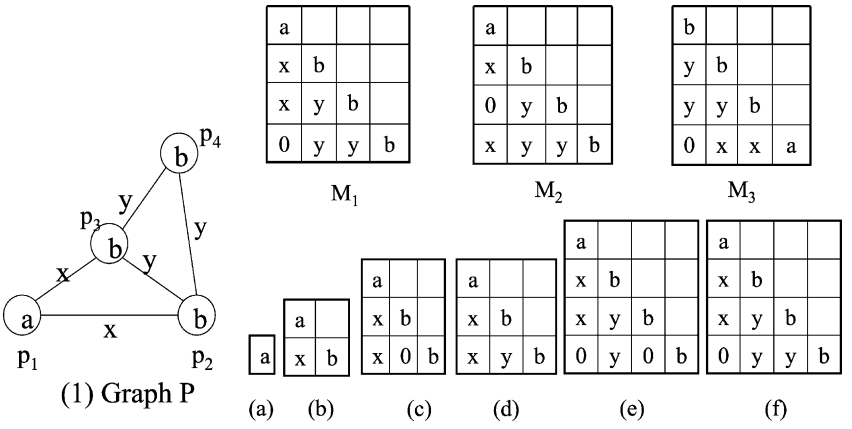

FIG. 11. Left: A labeled graph P. Upper right: Three adjacency matrices for the graph P. Lower right: Examples of maximal proper submatrices. Matrix (a) is the proper maximal submatrix of matrix (b), which itself is the proper maximal submatrix of (c) and so forth.

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 5.3. 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 Fig. 11. We notice that the empty string is a prefix of any string, and hence an empty matrix is the maximal proper submatrix of any matrix with size 1.

Definition 5.4. Given a graph space G^*, we define a binary relation $\preccurlyeq$ on graphs in 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 4. In Fig. 12, we have that $A \preccurlyeq B \preccurlyeq C \preccurlyeq D \preccurlyeq E \preccurlyeq F$ because of the maximal proper submatrix relation they have.

Theorem 1. *$\preccurlyeq$ is a partial order.*

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

- reflective: $G \preccurlyeq G$ for all graphs G,

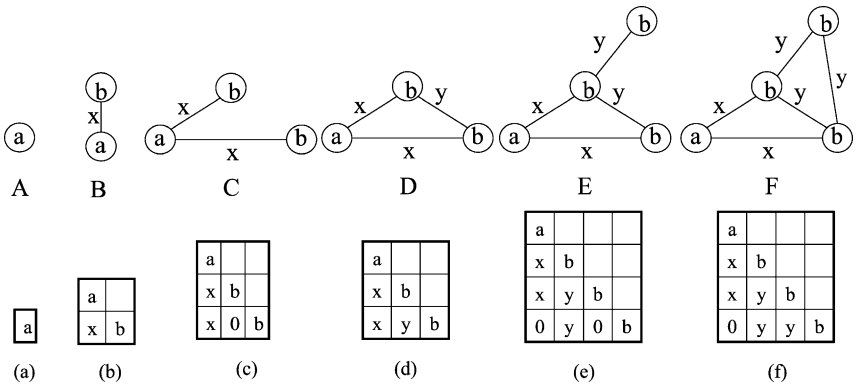

FIG. 12. Examples of the partial order $\preccurlyeq$. Upper: A group of graphs. Lower: The related CAM representations.

- 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 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 $G' \preccurlyeq G$.

We notice that each graph can have at most one maximal proper submatrix and hence has only one incoming edge. In other words, the directed graph we defined is acyclic. 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 Fig. 13 we show the CAM tree of all subgraphs of the graph P from Fig. 11.

The following theorem guarantees that the directed acyclic (DAG) graph $\mathcal{D}$ we constructed is a rooted tree.

Theorem 2. *The graph $\mathcal{D}$ we constructed in Section 5.2 is a rooted tree with the empty graph as its root.*

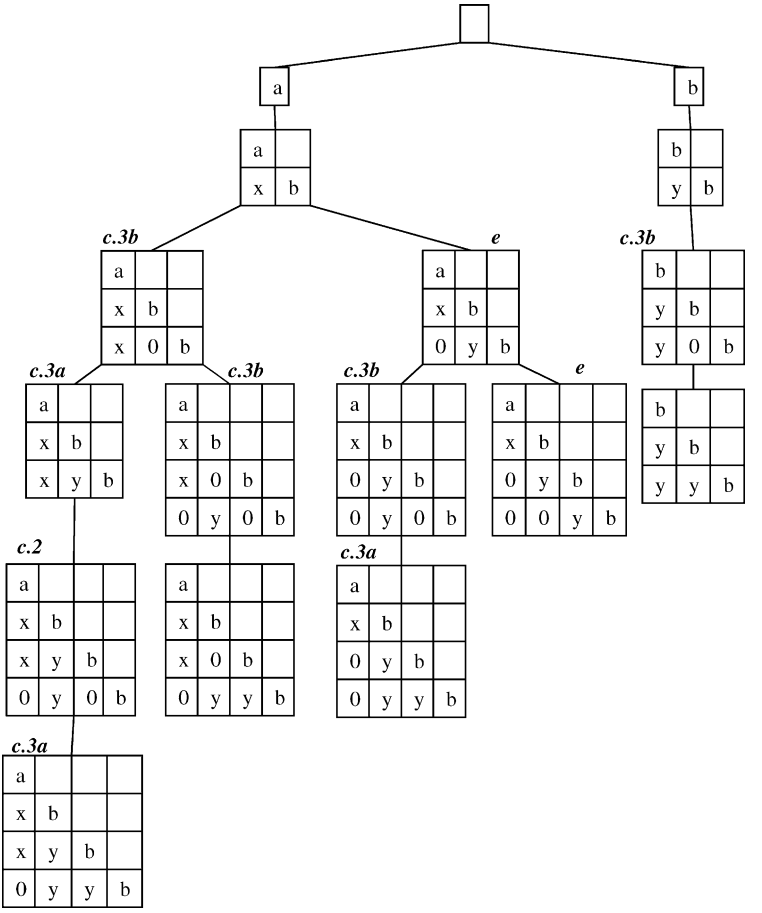

FIG. 13. The CAM Tree of all subgraphs of the graph P in Fig. 11. Every matrix obtained by a join operation is specified by a label starting with **c**. and then the type of the join operation e.g. **c.3a** stands for join case3a. A CAM obtained by an extension operation is labeled with **e**. The join and extension operations are discussed in Sections 5.3 and 5.4, respectively. CAMs (size $\geqslant 3$) without label are explained in Section 5.3 where suboptimal CAMs are discussed. CAMs with up to one edge are obtained by an initial step (discussed in Section 5.4) which involves directly scanning nodes/edges labels in a graph database.

Proof. We already have shown that $\mathcal{D}$ is a DAG. To prove that a DAG is a tree, all we need to do is to prove that for any graph G, there exists a sequence of graphs $G_1, G_2, \ldots, G_n$ such that G_1 is an empty graph, $G_n = G$ and $G_i \preccurlyeq G_{i+1}$ for $1 \leqslant i < n$. This is proved by the following theorem. □

Theorem 3. *Given a CAM M of a connected graph G and M's submatrix N, N represents a connected subgraph of G.*

Proof. 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 and 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. □

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 [45,55]. 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 [55], 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, [8,108] 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, we have augmented the CAM tree with a set of *sub-optimal canonical adjacency matrices*, and designed an embedding based subgraph enumeration method. Experimental evidence demonstrates our method can achieve an order of magnitude speed up over the current state-of-the-art subgraph mining algorithm gSpan [108]. Further details are discussed in the following sections.

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, let A's last edge be $a_{m,f}$ and B's last edge be $b_{n,k}$, and we define $join(A, B)$ by the following three cases:

join case 1: both A and B are inner matrices

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

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

$$c_{i,j} = \begin{cases} a_{i,j}, & 0 < i, j \leqslant m,\ i \neq n \text{ or } j \neq k, \\ b_{i,j}, & \text{otherwise.} \end{cases}$$

3: **else**

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

5: **end if**

join case 2: A is an inner matrix and B is an outer matrix $join(A, B) = \{C\}$ where C is a $n \times n$ matrix and

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

join case 3: both A and B are outer matrices

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

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

2: **if** $(f \neq k, a_{m,m} = b_{m,m})$ **then**
3: C is $m \times m$ matrix where (case 3a)

$$c_{i,j} = \begin{cases} a_{i,j}, & 0 < i, j \leqslant m, i \neq n \text{ or } j \neq k, \\ b_{i,j}, & \text{otherwise.} \end{cases}$$

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 $code(A) \geqslant code(B)$ 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 14 shows examples for the join operation for all four cases. At the bottom of Fig. 14, 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 Fig. 13. 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 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

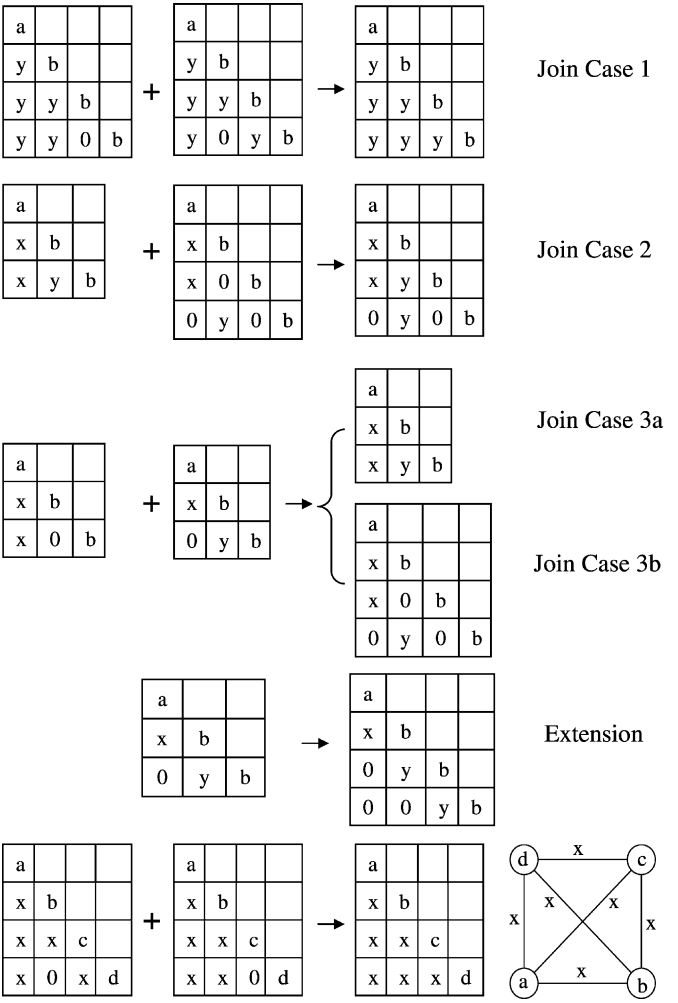

FIG. 14. Examples of the join/extension operation.

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 [108] 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

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

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

    $S \leftarrow S \cup \{B\}$
  end for
else
  $S \leftarrow \Phi$
end if

ALGORITHM 7. FFSM-Extension(A).

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. Here, we refer to [108] 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 4, this extension operation, combined with the join operation, unambiguously enumerates all the nodes in the CAM tree.

The pseudo code presenting the extension operation is shown in Algorithm 7.

5.3.3 Suboptimal CAM Tree

Using the CAM tree of the graph P in Fig. 13, 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 5.5. 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 N represents.

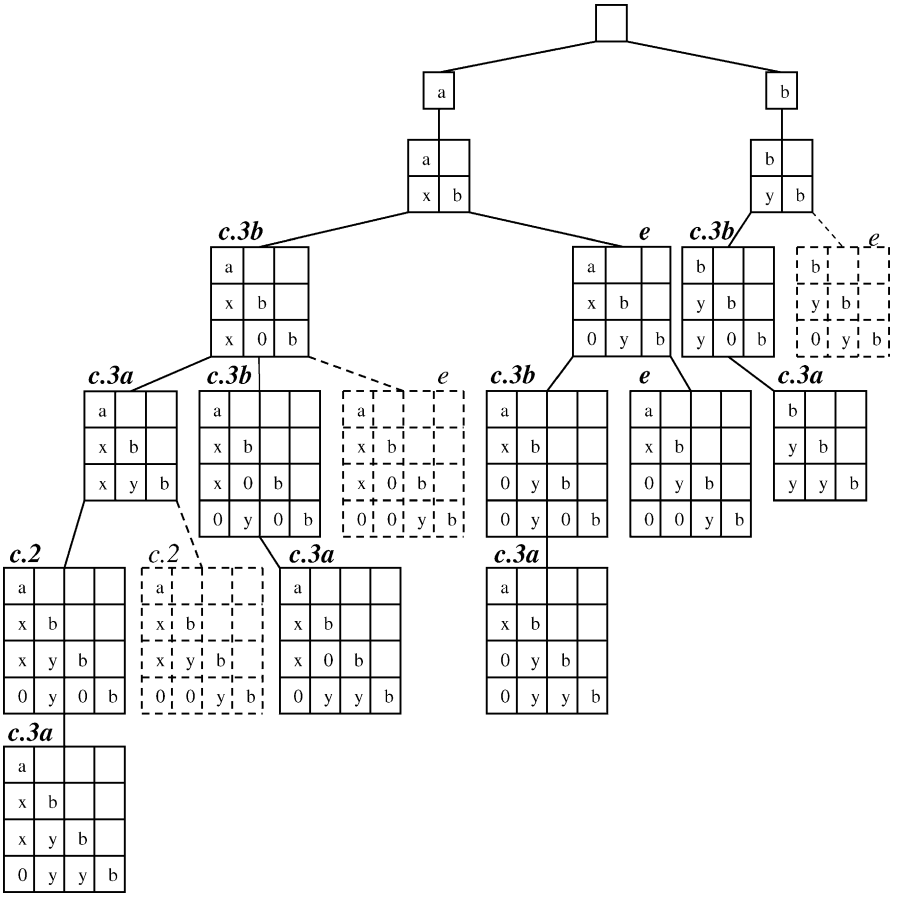

FIG. 15. The suboptimal CAM Tree for the graph P shown in Fig. 11. Matrices with solid boundary are CAMs and those with dashed line boundary are proper suboptimal CAMs. The label on top of an adjacency matrix M indicates the operation by which M might be proposed from its parent. The labeling follows the same conventions used in Fig. 13.

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 Fig. 15. 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 Fig. 11 is shown in Fig. 15.

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 4. *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 ($k \geqslant 3$). 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. 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 [40], 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. □

5.4 Mining Frequent Subgraphs

In the above discussions, we introduced a novel data structure (CAM tree) for organizing all connected subgraphs of a single connected undirected graph. This, however, can be easily extended to a set of graphs (connected or not), denoted as a graph database. 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), any traversal of the tree reveals 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 very unlikely scale to large graph databases.

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

ALGORITHM 8. FFSM($\mathcal{G}, \sigma$).

1: **for each** $X \in P$ **do**
2: **if** ($X.isCAM$) **then**
3: $F \leftarrow F \cup \{X\}$, $C \leftarrow \emptyset$
4: **for each** $Y \in P$ **do**
5: $C \leftarrow C \cup$ FFSM-Join(X, Y)
6: **end for**
7: $C \leftarrow C \cup$ FFSM-Extension(X)
8: $C \leftarrow \{G \mid G \in C, G \text{ is frequent}, G \text{ is suboptimal}\}$
9: $F \leftarrow F \cup$ FFSM-Explore(C, F)
10: **end if**
11: **end for**
12: **return** F

ALGORITHM 9. FFSM-explore(P, F).

In the following pseudo code, we present an algorithm which takes advantage of the following simple fact: if a subgraph G is not frequent (support of G is less than a user posted threshold), none of its supergraphs is frequent. This suggest 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 of Algorithms 8 and 9, symbol $\mathcal{M}(G)$ denotes the CAM of the 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 $\mathcal{M}(G)$).

5.5 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 2 GHz Pentium PC with 2 GB 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 [108].

5.5.1 Chemical Compound Data Sets

5.5.1.1 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 [90] 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 set. The atoms and bonds information are stored in two separate files and we follow exactly the same procedure described in [108] 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 42,115 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 respectively. The DTP datasets can be downloaded from http://dtp.nci.nih.gov/docs/aids/aids_data.html.

5.5.1.2 Performance Comparison. We evaluate the performance of FFSM using various support thresholds. The result is summarized in Figs. 16 and 17. We find that FFSM has a maximal 7 fold speedup over gSpan on the DTP CM data

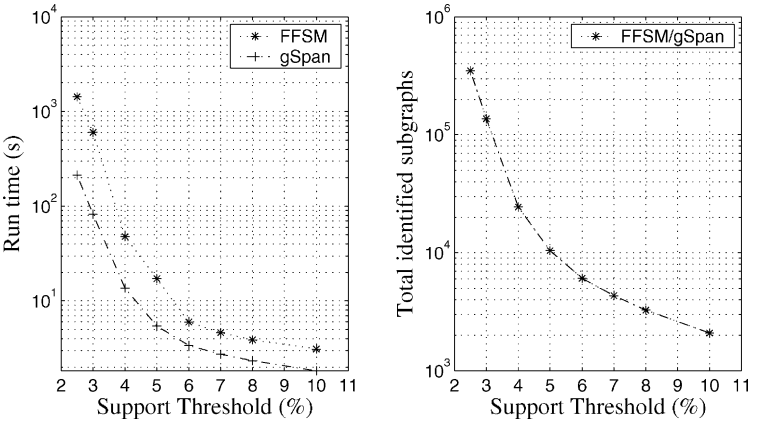

FIG. 16. Left: Performance comparison of FFSM and gSpan with different support values for the DTP CM data set. Right: The total number of frequent patterns identified by the algorithms.

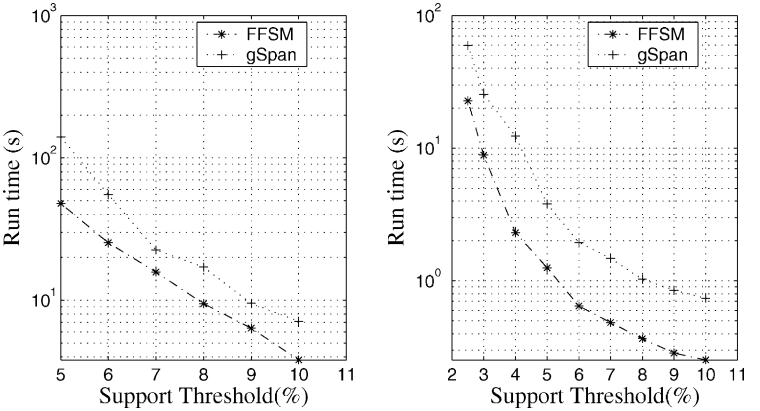

FIG. 17. Performance comparison of FFSM and gSpan with different support values for the DTP CA data set (left) and PTE (right).

set. For DTP CA and PTE data set, FFSM usually has a 2 to 3 fold speedup from gSpan.

5.5.2 Synthetic Data Sets

5.5.2.1 Data Sets. We used a graph generator offered by M. Kuramochi and G. Karypis [55] 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.

$$\text{“}D10kT20L200I9V4E4\text{”}$$

represents a synthetic graph database which contains a total of $|D| = 10$k (10,000) 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 $|I| = 9$.

5.5.2.2 Performance Comparison. In Fig. 18, we show how the FFSM algorithm scales with increasing support. The total number of identified frequent subgraphs is also given.

At the left part of Fig. 19, we show performance comparison between FFSM and gSpan with different average graph sizes (left) or different number of node/edge

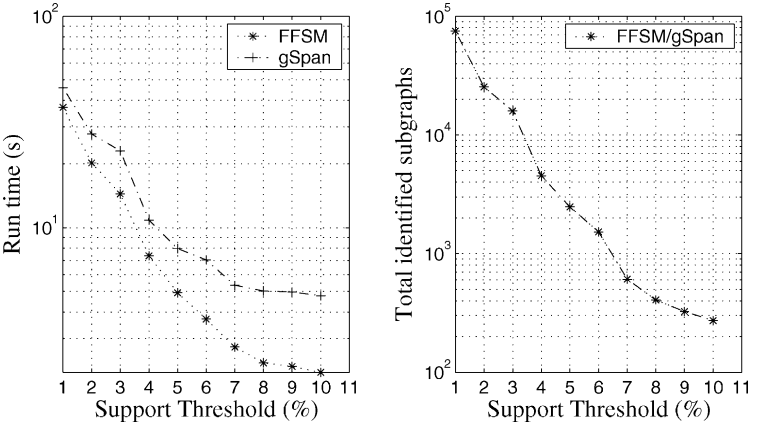

FIG. 18. FFSM and gSpan performance comparison under different support values. Parameters used: D10kT20I9L200E4V4.

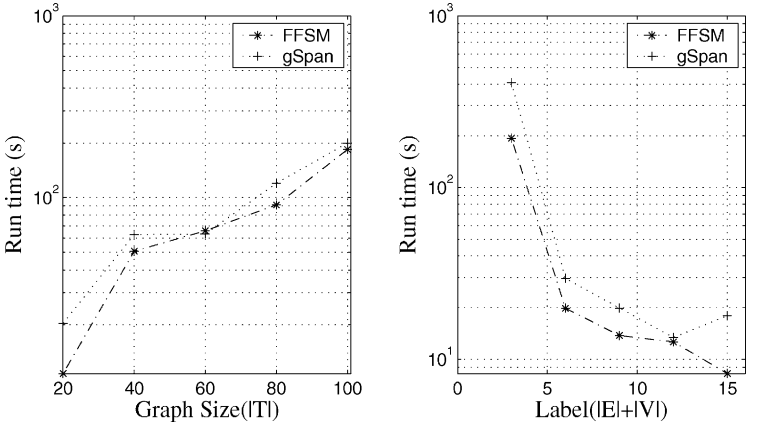

FIG. 19. FFSM and gSpan performance comparison under different graph sizes ($|T|$) ranging from 20 to 100 (left) or different total labels ($|V| + |E|$) ranging from 3 to 18 (right). The ratio of the $|V|$ to $|E|$ is fixed to 2 : 1 for any given total number of labels. For example, if there are total 15 labels, we have 10 vertex labels and 5 edge labels. Other parameters setting: D10kI7L200E4V4 (left) and D10kT20I7L200 (right). The support threshold is fixed at 1% in both cases.

labels (right). For almost all circumstances, FFSM is faster than gSpan though the value of the speedup varies from data set to data set.

5.5.3 Mining Protein Contact Graphs

5.5.3.1 Data Sets. We collect a group of serine proteases from the Structure Classification of Proteins database [62] 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 $\varepsilon > 0$ Å. The result is known as the almost-Delaunay edges for the point set [4]. We define that two residues are in *contact* if they are connected by an almost-Delaunay edges with $\varepsilon = 0.1$ 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 [39] and the data set is downloadable from http://www.cs.unc.edu/~huan/FFSM.shtml.

5.5.3.2 Performance Comparison. The current gSpan is specifically developed for small graphs (with no more than 200 edges in any graphs in a data set).

TABLE I
PERFORMANCE COMPARISON BETWEEN FFSM AND FSG

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

σ support threshold (percentage). Performance of FFSM and FSG are measured in seconds.

We compare FFSM with another graph mining algorithm FSG [55]. FFSM always an order of magnitude faster than FSG. Table I summarizes the results.

So far, we show the performance comparison between different graph mining algorithms. In the next section, we show how graph mining may be applied to protein structures to derive common structure patterns.

6. Applications

In this section we describe the use of the FFSM algorithm presented in Section 5 to identify family-specific structural motifs for a number of protein families.

6.1 Identifying Structure Motifs

6.1.1 Representing Protein Structure As a Labeled Graph

We model protein structure as a labeled graph where a node represents an amino acid residue, labeled by the amino acid identity, and an edge joins a pair of amino acids, labeled by the Euclidian distance between two C_α atoms. To reduce complexity, we eliminate edges with distances larger than 12.5 Å [23,107]. We partition the one-dimensional distance space into bins in order to tolerate position uncertainty. The width of such bins is referred to as the *distance tolerance* and popular choices are 1 Å [61], 1.5 Å [9], and 2 Å [79]. We use 1.5 Å exclusively in our experimental study.

Given the graph representation, a recurring pattern may be composed of points with no possible physical and chemical interactions among them. This distributed set of points, though geometrically conserved, is hard to assign any biological interpretation to and is usually considered uninteresting by domain experts. To avoid spending computational resources on such patterns, we designate a subset of edges as *contacts* where a contact is an edge joining a pair of points (amino acids) that we believe may interact with each other (as described below). We require that each pattern is a connected component with respect to the contact edges. Similar strategies are used to derive structural patterns with high quality by others [59].

6.1.1.1 Defining Contacts of Amino Acid Residues. There are many ways to define whether two amino acids are in contact or not. In our study, two points are in *contact* if they can be connected by a Delaunay edge [88] with point coordinates perturbation up to $\varepsilon \geqslant 0$. Such Delaunay edges (with point coordinate perturbations) are extensions of the commonly used Delaunay edges that are defined on static points [4]. We further restrict the contact edges to have distances no greater

than some upper limit ranging from 6.5 to 8.5 Å; this value represents an upper limit on the distance over which there can be significant interaction between amino acid residues.

The graph model presented here is similar to that used by other groups [77,104]. The major difference is that in our representation, geometric constraints such as distances between amino acids are part of the graph representation in order to obtain geometrically conserved patterns rather than using a loosely constrained graph, to reduce the number of spurious patterns.

6.1.2 Graph Database Mining

We apply the FFSM algorithm to find recurring patterns from protein structures. To enforce maximal geometric constraints, we only report fully connected subgraph (i.e. cliques) with all inter-residue distances specified. In graph matching, we require that matching nodes have the same label and matching edges have the same label and type (contact or not). Enforcing these, we guarantee that the structural patterns reported by our system have well defined composition of amino acid identity and three dimensional shape.

6.1.3 Statistical Significance of Motifs

We derived an empirical evaluation of the statistical significance of structural patterns. We randomly sampled proteins from the protein structure space and applied our pattern mining algorithm to search for patterns. The experiments were repeated many times to estimate the probability that we observe at least one pattern using randomly selected proteins. The lower this probability is, the higher confidence we have about the significance of any structural patterns that are found among a group of proteins.

6.1.3.1 Estimating Significance by Random Sampling. In our experimental study, we randomly sampled 20 proteins (without replacement) from an non-redundant PDB list [102] and applied our algorithm to search for patterns with support $\geqslant 15$ and with pattern size of at least 4 amino acid residues. These parameters were set up to mimic a typical size and search 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.

6.1.3.2 Estimating Significance using the 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 used the structures in the Culled PDB list [102], as a set of structures M that sample the entire protein structure population (all possible protein structures, crystallized 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 [9]:

$$P\text{-}value = 1 - \sum_{i=0}^{k-1} \frac{\binom{|F|}{i}\binom{|M|-|F|}{|T|-i}}{\binom{|M|}{|T|}} \tag{2}$$

where $|X|$ is the cardinality of a set X. For example, if a pattern m occurs in every member of a family F and never 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 [82]: $0.001/|C|$, where $|C|$ is the set of categories. The correction is used as the threshold for significance of the P-value of an individual test. Since the total number of SCOP families is 2327, a significant P-value is $\leqslant 10^{-7}$.

6.2 Case Studies

As a proof-of-concept, we applied the method to identify family-specific motifs, i.e. structural patterns that occur frequently in a family and rarely outside it. In Table II, 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.

6.2.1 Eukaryotic Serine Proteases

The structural patterns identified from the ESP family were documented at the top part of Table II. The data indicated that the patterns we found are highly specific to the ESP family, measured by $P\text{-}value \leqslant 10^{-82}$. 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 Fig. 20. 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.

TABLE II
STRUCTURAL PATTERNS IDENTIFIED IN THE EUKARYOTIC SERINE PROTEASE, PAPAIN-LIKE CYSTEINE PROTEASE, AND NUCLEAR BINDING DOMAINS

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	AGGG	50	58	85
2	ACGG	52	9	100	21	ACGAG	49	4	100
3	DHSC	52	10	100	22	SCGA	49	6	100
4	DHSA	52	10	100	23	DACS	49	7	100
5	DSAC	52	12	100	24	DGGS	49	8	100
6	DGGG	52	23	100	25	SACG	49	10	98
7	DHSAC	51	9	100	26	DSGC	49	15	98
8	SAGC	51	11	100	27	DASC	49	20	92
9	DACG	51	14	100	28	SAGG	49	31	90
10	HSAC	51	14	100	29	DGGL	49	53	83
11	DHAA	51	18	100	30	DSAGC	48	9	99
12	DAAC	51	32	99	31	DSSC	48	12	97
13	DHAAC	50	5	100	32	SCSG	48	19	93
14	DHAC	50	6	100	33	AGAG	48	19	93
15	HACA	50	8	100	34	SAGG	48	23	88
16	ACGA	50	11	100	35	DSGS	48	23	94
17	DSAG	50	16	100	36	DAAG	48	27	89
18	SGGC	50	17	100	37	DASG	48	32	87
19	AGAG	50	27	95	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	WGNS	18	4	44
2	HCQG	18	3	34	5	WGSG	18	5	43
3	WWGS	18	3	44					
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

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 seconds). 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 Eq. (2). 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 6.2.

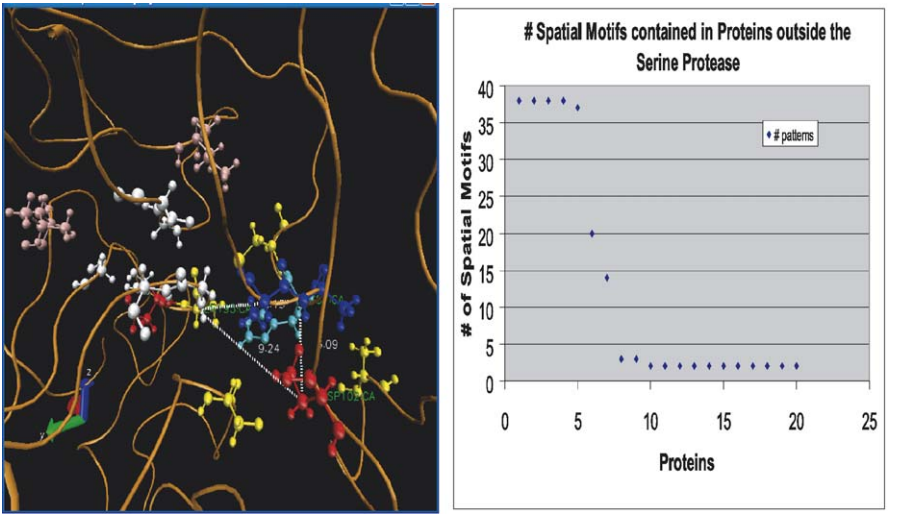

FIG. 20. Left: Spatial distribution of residues found in 38 common structural patterns within protein 1HJ9. The residues of catalytic triad, HIS57-ASP102-SER195, are connected by white dotted lines. Right: Instances of ESP structural patterns occurring in proteins outside the ESP data set. The top 7 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 chapter 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 found seven proteins: 1pq7a, 1os8a, 1op0a, 1p57b, 1s83a, 1ssxa, and 1md8a, that contain quite a few patterns, as shown in Fig. 20. 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 [104].

6.2.2 *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 II.

For the 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 [14], as shown in Fig. 21. For the NB family, we identified four patterns[3] which map to the cofactor binding sites [103], shown in the same figure. In addition, four members missed by SCOP: 1srv, 1khq, and 1o0e were identified for

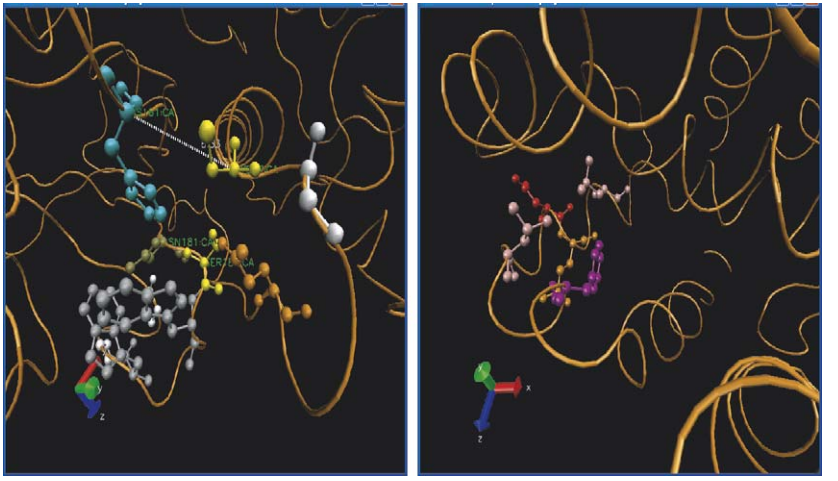

FIG. 21. Left: Residues included in the patterns from PCP family in protein 1CQD. The residues in catalytic dyad CYS27-HIS161 are connected by a white dotted line and two important surrounding residues ASN181 and SER182 are labeled. Right: Residues included in patterns from the NB family in protein 1OVL. The labeled residue GLN 435 has direct interaction with the cofactor of the protein.

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

the PCP family and six members 1sj0, 1rkg, 1osh, 1nq7, 1pq9, 1nrl were identified for the NB family.

6.2.3 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 FFSM to the largest family in the 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 significance in the NAD(P)-binding Rossmann-fold superfamily as shown in bottom part of Table II.

In Fig. 22, 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 significant common features can be inferred from proteins with no apparent sequence and fold similarity.

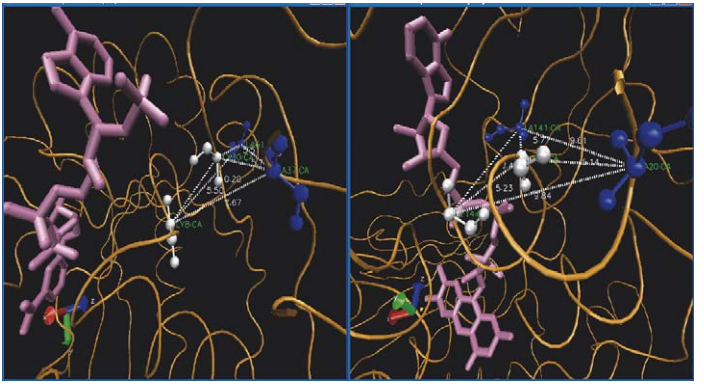

FIG. 22. The pattern appears in two proteins 1LVL (belongs to the FAD/NAD-linked reductase family without Rossman fold) and 1JAY (belongs to the 6-phosphogluconate dehydrogenase-like, N-terminal domain family with Rossman fold) with conserved geometry.

7. Conclusions and Future Directions

7.1 Conclusions

Structure comparison of proteins is 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 to domains 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 efforts will rapidly produce a large volume of structure and structure-related data.

Below, we review plausible future directions that we think are important for structure comparison.

7.2 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.

7.2.1 Future Applications of Structural Comparison

Three future applications of structure comparison are discussed.

7.2.1.1 Understanding 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 [106].

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 [72]. 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 [106,44].

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, we believe that such techniques, as well as computational methods for molecular dynamics, will mature rapidly and be successful in helping domain experts gain useful insights into various biological processes.

7.2.1.2 *Predicting 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 [1].

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.

7.2.1.3 *Predicting Protein 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 [25]. As another example, knowing the localization of a protein offers important information for assembling metabolic pathways [80].

Predicting the subcellular localization is one of the active research topics in bioinformatics research [25,64,80]. 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 moti-

vated by the observation that residue composition of a protein highly correlates with the localization of the proteins [64].

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 significant improvement of the prediction accuracy, as reported in [25]. 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 [33]. As shown in this preliminary study, incorporating structure comparison can improve the accuracy of the protein subcellular prediction.

7.2.2 New Computational Techniques in Structure Comparison

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 structures 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.

We conclude this chapter with a brief description of statistical structure comparison.

7.2.2.1 Comparison Based on 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 [2] 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 [74] to find the best alignment of the query structure to the HMM model, and
(4) using the Forward algorithm [74] 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 [7], Hidden Markov Random Field, and Bayesian Networks [43]. It will be interesting to see their applicability in protein structure comparison.

REFERENCES

[1] Aebersold R., Mann M., "Mass spectrometry-based proteomics", *Nature* **422** (March 13, 2003) 198–207.
[2] Alexandrov V., Gerstein M., "Using 3d hidden Markov models that explicitly represent spatial coordinates to model and compare protein structures", *BMC Bioinformatics* **9** (5) (2004).
[3] Artymiuk P.J., Poirrette A.R., Grindley H.M., Rice D.W., Willett P., "A graph-theoretic approach to the identification of three-dimensional patterns of amino acid side-chains in protein structures", *J. Mol. Biol.* **243** (1994) 327–344.
[4] Bandyopadhyay D., Snoeyink J., "Almost-Delaunay simplices: Nearest neighbor relations for imprecise points", in: *ACM–SIAM Symposium on Distributed Algorithms*, 2004, pp. 403–412.
[5] Barker J.A., Thornton J.M., "An algorithm for constraint-based structural template matching: Application to 3d templates with statistical analysis", *Bioinformatics* **19** (13) (2003) 1644–1649.
[6] Berman H.M., Westbrook J., Feng Z., Gilliland G., Bhat T.N., Weissig H., Shindyalov I.N., Bourne P.E., "The protein data bank", *Nucl. Acids Res.* **28** (2000) 235–242.
[7] Besag J., "Spatial interaction and the statistical analysis of lattice systems", *J. Royal Statist. Soc. B* **36** (1974) 192–236.
[8] Borgelt C., Berhold M.R., "Mining molecular fragments: Finding relevant substructures of molecules", in: *Proc. International Conference on Data Mining'02*, 2002, pp. 51–58.
[9] Bradley P., Kim P.S., Berger B., "TRILOGY: Discovery of sequence-structure patterns across diverse proteins", *Proc. Natl. Acad. Sci.* **99** (13) (June 2002) 8500–8505.
[10] Branden C., Tooze J., *Introduction to Protein Structure*, Garland Publishing, New York, 1991.

[11] Burdick D., Calimlim M., Gehrke J., "Mafia: A maximal frequent itemset algorithm for transactional databases", in: *ICDE*, 2001.
[12] Cammer S.A., Carter C.W., Tropsha A., "Identification of sequence-specific tertiary packing motifs in protein structures using Delaunay tessellation", *Lecture Notes in Comput. Sci. Engrg.* **24** (2002) 477–494.
[13] Chance M.R., Bresnick A.R., Burley S.K., Jiang J.S., Lima C.D., Sali A., Almo S.C., Bonanno J.B., Buglino J.A., Boulton S., Chen H., Eswar N., He G., Huang R., Ilyin V., McMahan L., Pieper U., Ray S., Vidal M., Wang L.K., "Structural genomics: A pipeline for providing structures for the biologist", *Protein Sci.* **11** (2002) 723–738.
[14] Choi K.H., Laursen R.A., Allen K.N., "The 2.1 angstrom structure of a cysteine protease with proline specificity from ginger rhizome, zingiber officinale", *Biochemistry* **38** (36) (September 7, 1999) 11624–11633.
[15] Cordes M.H., Sauer R.T., "Tolerance of a protein to multiple polar-to-hydrophobic surface substitutions", *Protein Sci.* **8** (2) (1999) 318–325.
[16] Dehaspe L., Toivonen H., King R.D., "Finding frequent substructures in chemical compounds", in: *4th International Conference on Knowledge Discovery and Data Mining*, 1998, pp. 30–36.
[17] Deutsch A., Fernandez M.F., Suciu D., "Storing semistructured data with STORED", in: *SIGMOD*, 1999, pp. 431–442.
[18] D'haeseleer P., Liang S., Somogyi R., "Genetic network inference: From co-expression clustering to reverse engineering", *Bioinformatics* **16** (8) (2000) 707–726.
[19] Dodson G., Wlodawer A., "Catalytic triads and their relatives", *Trends Biochem. Sci.* **23** (9) (September 1998) 347–352.
[20] Dokholyan N.V., Buldyrev S.V., Stanley H.E., Shakhnovich E.I., "Identifying the protein folding nucleus using molecular dynamics", *J. Mol. Biol.* **296** (2000) 1183–1188.
[21] Eidhammer I., Jonassen I., Taylor W.R., *Protein Bioinformatics: An Algorithmic Approach to Sequence and Structure Analysis*, John Wiley & Sons, Ltd, New York, 2004.
[22] Fersht A., *Structure and Mechanism in Protein Science*, W.H. Freeman Co., New York, 1999.
[23] Fischer D., Wolfson H., Lin S.L., Nussinov R., "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 Sci.* **3** (1994) 769–778.
[24] Gardiner E.J., Artymiuk P.J., Willett P., "Clique-detection algorithms for matching three-dimensional molecular structures", *J. Mol. Graph. Model.* **15** (1997) 245–253.
[25] Gardy J.L., Spencer C., Wang K., Ester M., Tusnady G.E., Simon I., Hua S., deFays K., Lambert C., Nakai K., Brinkman F.S., "Psort-b: Improving protein subcellular localization prediction for gram-negative bacteria", *Nucleic Acids Res.* **31** (13) (2003) 3613–3617.
[26] George R.A., Spriggs R.V., Bartlett G.J., Gutteridge A., MacArthur M.W., Porter C.T., Al-Lazikani B., Thornton J.M., Swindells M.B., "Effective function annotation through residue conservation", *Proc. Natl. Acad. Sci.* **102** (2005) 12299–12304.
[27] George R.A., Spriggs R.V., Thornton J.M., Al-Lazikani B., Swindells M.B., "Scopec: A database of protein catalytic domains Supp", *Bioinformatics* (Suppl. 1) (2004) I130–I136.

[28] Gerlt J.A., Babbitt P.C., "Divergent evolution of enzymatic function: Mechanistically diverse superfamilies and functionally distinct suprafamilies", *Annu. Rev. Biochem.* **70** (2001) 20946.

[29] Goldman R., Widom J., "Dataguides: Enabling query formulation and optimization in semistructured databases", in: *VLDB'97*, 1997.

[30] Grindley H.M., Artymiuk P.J., Rice D.W., Willett P., "Identification of tertiary structure resemblance in proteins using a maximal common subgraph isomorphism algorithm", *J. Mol. Biol.* **229** (1993) 707–721.

[31] Hegyi H., Gerstein M., "The relationship between protein structure and function: A comprehensive survey with application to the yeast genome", *J. Mol. Biol.* **288** (1999) 147–164.

[32] Hermjakob H., Montecchi-Palazzi L., Lewington C., Mudali S., Kerrien S., Orchard S., Vingron M., Roechert B., Roepstorff P., Valencia A., Margalit H., Armstrong J., Bairoch A., Cesareni G., Sherman D., Apweiler R., "Intact—an open source molecular interaction database", *Nucl. Acids Res.* **32** (2004) D452–D455.

[33] Himukai R., Kuzuhara T., Horikoshi M., "Relationship between the subcellular localization and structures of catalytic domains of fkbp-type ppiases", *J. Biochem. (Tokyo)* **126** (5) (1999) 879–888.

[34] Holder L.B., Cook D.J., Djoko S., "Substructures discovery in the subdue system", in: *Proc. AAAI'94 Workshop Knowledge Discovery in Databases*, 1994, pp. 169–180.

[35] Holm L., Sander C., "Mapping the protein universe", *Science* **273** (1996) 595–602.

[36] Horn B.K.P., "Closed-form solution of absolute orientation using unit quaternions", *J. Opt. Soc. Amer. A: Opt. Image Sci. Vision* **4** (4) (1987) 629–642.

[37] Hu J., Shen X., Shao Y., Bystroff C., Zaki M.J., "Mining protein contact maps", in: *2nd BIOKDD Workshop on Data Mining in Bioinformatics*, 2002.

[38] Huan J., Bandyopadhyay D., Wang W., Snoeyink J., Prins J., Tropsha A., "Comparing graph representations of protein structure for mining family-specific residue-based packing motifs", *J. Comput. Biol.* **12** (6) (2005) 657–671.

[39] Huan J., Wang W., Bandyopadhyay D., Snoeyink J., Prins J., Tropsha A., "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*, 2004, pp. 308–315.

[40] Huan J., Wang W., Prins J., "Efficient mining of frequent subgraph in the presence of isomorphism", in: *Proceedings of the 3rd IEEE International Conference on Data Mining, ICDM*, 2003, pp. 549–552.

[41] Huan J., Wang W., Prins J., Yang J., "SPIN: Mining maximal frequent subgraphs from graph databases", in: *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2004, pp. 581–586.

[42] Huan J., Wang W., Washington A., Prins J., Shah R., Tropsha A., "Accurate classification of protein structural families based on coherent subgraph analysis", in: *Proceedings of the Pacific Symposium on Biocomputing, PSB*, 2004, pp. 411–422.

[43] Huang C., Darwiche A., "Inference in belief networks: A procedural guide", *Internat. J. Approx. Reasoning* **15** (3) (1996) 225–263.

[44] Hubbell W.L., Cafiso D.S., Altenbach C., "Identifying conformational changes with site-directed spin labeling", *Natl. Struct. Biol.* **7** (9) (2000) 735–739.
[45] Inokuchi A., Washio T., Motoda H., "An apriori-based algorithm for mining frequent substructures from graph data", in: *PKDD'00*, 2000, pp. 13–23.
[46] Jonassen I., "Efficient discovery of conserved patterns using a pattern graph", *Comput. Appl. Biosci.* **13** (5) (1997) 509–522.
[47] Jonassen I., Eidhammer I., Conklin D., Taylor W.R., "Structure motif discovery and mining the PDB", *Bioinformatics* **18** (2002) 362–367.
[48] Jonassen I., Eidhammer I., Taylor W.R., "Discovery of local packing motifs in protein structures", *Proteins* **34** (1999) 206–219.
[49] Jones S., Thornton J.M., "Searching for functional sites in protein structures", *Curr. Opin. Chem. Biol.* **8** (2004) 3–7.
[50] Kabsch W.A., "Discussion of solution for best rotation of two vectors", *Acta Crystallogr. A* **34** (1978) 827–828.
[51] Kelley L.A., MacCallum R.M., Sternberg M.J., "Enhanced genome annotation using structural profiles in the program 3d-pssm", *J. Mol. Biol.* **299** (2) (2000) 499–520.
[52] Kendrew J.C., Bodo G., Dintzis H.M., Parrish R.G., Wyckoff H., Phillips D.C., "A three-dimensional model of the myoglobin molecule obtained by X-ray analysis", *Nature* **181** (1958) 662–666.
[53] Koonin E.V., Wolf Y.I., Karev G.P. (Eds.), *Power Laws, Scale-Free Networks and Genome Biology*, Springer-Verlag, Berlin, 2004.
[54] Krishnamoorthy B., Tropsha A., "Development of a four-body statistical pseudo-potential to discriminate native from non-native protein conformations", *Bioinformatics* **19** (12) (2003) 1540–1548.
[55] Kuramochi M., Karypis G., "Frequent subgraph discovery", in: *Proc. International Conference on Data Mining'01*, 2001, pp. 313–320.
[56] Leibowitz N., Fligelman Z.Y., Nussinov R., Wolfson H.J., "Automated multiple structure alignment and detection of a common substructural motif", *Proteins* **43** (3) (May 2001) 235–245.
[57] Lupasa A.N., Pontingb C.P., Russell R.B., "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?", *J. Struct. Biol.* **134** (2001) 191–203.
[58] Matthews B.W., "Structural and genetic analysis of the folding and function of t4 lysozyme", *FASEB J.* **10** (1996) 35–41.
[59] Coatney M., Parthasarathy S., "Motifminer: A toolkit for mining common substructures in molecular data", *Knowledge Inform. Syst. J.* (2003).
[60] Meng E.C., Polacco B.J., Babbitt P.C., "Superfamily active site templates", *Proteins* **55** (4) (2004) 962–976.
[61] Milik M., Szalma S., Olszewski K.A., "Common structural cliques: A tool for protein structure and function analysis", *Protein Engrg.* **16** (8) (2003) 543–552.
[62] Murzin A.G., Brenner S.E., Hubbard T., Chothia C., "SCOP: A structural classification of proteins database for the investigation of sequences and structures", *J. Mol. Biol.* **247** (1995) 536–540.

[63] Nagano N., Orengo C.A., Thornton J.M., "One fold with many functions: The evolutionary relationships between TIM barrel families based on their sequences, structures and functions", *J. Mol. Biol.* **321** (2002) 741–765.
[64] Nakashima H., Nishikawa K., "Discrimination of intracellular and extracellular proteins using amino acid composition and residue-pair frequencies", *J. Mol. Biol.* **238** (1) (1994) 54–61.
[65] Neidhart D.J., Kenyon G.L., Gerlt J.A., Petsko G.A., "Mandelate racemase and muconate lactonizing enzyme are mechanistically distinct and structurally homologous", *Nature* **347** (1990) 692–694.
[66] Nijssen S., Kok J.N., "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*, 2004, pp. 647–652.
[67] Nussinov R., Wolfson H.J., "Efficient detection of three-dimensional structural motifs in biological macromolecules by computer vision techniques", *Proc. Natl. Acad. Sci.* **88** (1991) 10495–10499.
[68] Orengo C.A., Michie A.D., Jones S., Jones D.T., Swindells M.B., Thornton J.M., "CATH—a hierarchic classification of protein domain structures", *Structure* **5** (8) (1997) 1093–1108.
[69] Orgengo C., Jones D., Thornton J., *Bioinformatics: Genes, Proteins, and Computers*, BIOS Scientific Publishers Ltd, 2003.
[70] Overbeek R., Fonstein M., D'Souza M., Pusch G.D., Maltsev N., "The use of gene clusters to infer functional coupling", *Proc. Natl. Acad. Sci.* **96** (6) (1999) 2896–2901.
[71] Pennec X., Ayache N., "A geometric algorithm to find small but highly similar 3d substructures in proteins", *Bioinformatics* **14** (6) (1998) 516–522.
[72] Petsko G.A., Ringe D., *Protein Structure and Function*, New Science Press Ltd, Middlesec House, 34–42 Cleveland Street, London W1P 6LB, UK, 2004.
[73] Phizicky E., Bastiaens P.I.H., Zhu H., Snyder M., Fields S., "Protein analysis on a proteomic scale", *Nature* **422** (March 13, 2003) 208–215.
[74] Rabiner L.R., Juang B.H., "An introduction to hidden Markov models", *IEEE ASSP Magazine* (January 1986) 4–15.
[75] Raghavan S., Garcia-Molina H., "Representing web graphs", in: *Proceedings of the IEEE International Conference on Data Engineering*, 2003.
[76] Richardson J.S., "Class-directed structure determination: Foundation for a protein structure initiative", *Adv. Protein Chem.* **34** (1981) 167–339.
[77] Russell R.B., "Detection of protein three-dimensional side-chain patterns: New examples of convergent evolution", *J. Mol. Biol.* **279** (1998) 1211–1227.
[78] Sander C., Schneider R., "Database of homology-derived protein structures and the structural meaning of sequence alignment", *Proteins* **9** (1) (1991) 56–68.
[79] Schmitt S., Kuhn D., Klebe G., "A new method to detect related function among proteins independent of sequence and fold homology", *J. Mol. Biol.* **323** (2) (2002) 387–406.
[80] Schneider G., Fechner U., "Advances in the prediction of protein targeting signals", *Proteomics* **4** (6) (June 2004) 1571–1580.
[81] Schwehm J.M., Kristyanne E.S., Biggers C.C., Stites W.E., "Stability effects of increasing the hydrophobicity of solvent-exposed side chains in staphylococcal nuclease", *Biochemistry* **37** (19) (1998) 6939–6948.

[82] Shaffer J.P., "Multiple hypothesis testing", *Annu. Rev. Psychol.* (1995) 561–584.
[83] Sharan R., Suthram S., Kelley R.M., Kuhn T., McCuine S., Uetz P., Sittler T., Karp R.M., Ideker T., "Conserved patterns of protein interaction in multiple species", *Proc. Natl. Acad. Sci.* **102** (6) (2005) 1974–1979.
[84] Sharan R., Ideker T., Kelley B.P., Shamir R., Karp R.M., "Identification of protein complexes by comparative analysis of yeast and bacterial protein interaction data", in: *ACM RECOMB*, 2004, pp. 282–289.
[85] Shatsky M., Shulman-Peleg A., Nussinov R., Wolfson H.J., "Recognition of binding patterns common to a set of protein structures, in: *RECOMB*, 2005, submitted for publication.
[86] Shearer K., Bunks H., Venkatesh S., "Video indexing and similarity retrieval by largest common subgraph detection using decision trees", *Pattern Recogn.* **34** (5) (2001) 1075–1091.
[87] Shulman-Peleg A., Nussinov R., Wolfson H.J., "Recognition of functional sites in protein structures", *J. Mol. Biol.* **339** (3) (June 2004) 607–633.
[88] Singh R.K., Tropsha A., Vaisman I.I., "Delaunay tessellation of proteins", *J. Comput. Biol.* **3** (1996) 213–222.
[89] Spriggs R.V., Artymiuk P.J., Willett P., "Searching for patterns of amino acids in 3D protein structures", *J. Chem. Inform. Comput. Sci.* **43** (2003) 412–421.
[90] Srinivasan A., King R.D., Muggleton S.H., Sternberg M., "The predictive toxicology evaluation challenge", in: *Proc. of the 15th International Joint Conference on Artificial Intelligence, IJCAI*, 1997, pp. 1–6.
[91] Stark A., Russell R.B., "Annotation in three dimensions. Pints: Patterns in non-homologous tertiary structures", *Nucl. Acids Res.* **31** (13) (2003) 3341–3344.
[92] Stark A., Shkumatov A., Russell R.B., "Finding functional sites in structural genomics proteins", *Structure (Camb)* **12** (2004) 1405–1412.
[93] Stark A., Sunyaev S., Russell R.B., "A model for statistical significance of local similarities in structure", *J. Mol. Biol.* **326** (1998) 1307–1316.
[94] Terwilliger T.C., Waldo G., Peat T.S., Newman J.M., Chu K., Berendzen J., "Class-directed structure determination: Foundation for a protein structure initiative", *Protein Sci.* **7** (1998) 1851–1856.
[95] Torrance J.W., Bartlett G.J., Porter C.T., Thornton J.M., "Using a library of structural templates to recognise catalytic sites and explore their evolution in homologous families", *J. Mol. Biol.* **347** (2005) 565–581.
[96] Tropsha A., Carter C.W., Cammer S., Vaisman I.I., "Simplicial neighborhood analysis of protein packing (SNAPP): A computational geometry approach to studying proteins", *Methods Enzymol.* **374** (2003) 509–544.
[97] Ullman J.D., "An algorithm for subgraph isomorphism", *J. Assoc. Comput. Machinery* **23** (1976) 31–42.
[98] Vanetik N., Gudes E., Shimony E., "Computing frequent graph patterns from semi-structured data", in: *Proc. International Conference on Data Mining'02*, 2002.
[99] Via A., Ferre F., Brannetti B., Valencia A., Helmer-Citterich M., "Three-dimensional view of the surface motif associated with the p-loop structure: cis and trans cases of convergent evolution", *J. Mol. Biol.* **303** (4) (November 2000) 455–465.

[100] von Mering C., Huynen M., Jaeggi D., Schmidt S., Bork P., Snel B., "String: A database of predicted functional associations between proteins", *Nucl. Acids Res.* **31** (2003) 258–261.

[101] Wallace A.C., Borkakoti N., Thornton J.M., "Tess: A geometric hashing algorithm for deriving 3d coordinate templates for searching structural databases. Application to enzyme active sites", *Protein Sci.* **6** (11) (1997) 2308–2323.

[102] Wang G., Dunbrack R.L., "PISCES: A protein sequence culling server", *Bioinformatics* **19** (2003) 1589–1591;
http://www.fccc.edu/research/labs/dunbrack/pisces/culledpdb.html.

[103] Wang Z., Benoit G., Liu J., Prasad S., Aarnisalo P., Liu X., Xu H., Walker N.P., Perlmann T., "Structure and function of nurr1 identifies a class of ligand-independent nuclear receptors", *Nature* **423** (3) (2003) 555–560.

[104] Wangikar P.P., Tendulkar A.V., Ramya S., Mali D.N., Sarawagi S., "Functional sites in protein families uncovered via an objective and automated graph theoretic approach", *J. Mol. Biol.* **326** (3) (2003) 955–978.

[105] Weir M., Swindells M., Overington J., "Insights into protein function through large-scale computational analysis of sequence and structure", *Trends Biotechnol.* **19** (Suppl. 10) (2001) s61–s66.

[106] Weiss S., "Measuring conformational dynamics of biomolecules by single molecule fluorescence spectroscopy", *Nature Struct. Biol.* **7** (9) (2000) 724–729.

[107] Weskamp N., Kuhn D., Hullermeier E., Klebe G., "Efficient similarity search in protein structure databases by k-clique hashing", *Bioinformatics* **20** (2004) 1522–1526.

[108] Yan X., Han J., "gspan: Graph-based substructure pattern mining", in: *Proc. International Conference on Data Mining'02*, 2002, pp. 721–724.

[109] Yan X., Han J., "Closegraph: Mining closed frequent graph patterns", in: *KDD'03*, 2003.

[110] Zaki M.J., "Efficiently mining frequent trees in a forest", in: *SIGKDD'02*, 2002.