Protein Structure Prediction by Linear Programming

by

Jinbo Xu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Doctor of Philosophy
in
Computer Science

Waterloo, Ontario, August 2003

I hereby declare that I am the sole author of this thesis.
I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Jinbo Xu

I authorize the University of Waterloo to reproduce this thesis by photocopying or other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Jinbo Xu

The University of Waterloo requires the signatures of all persons using or photocopying this thesis. Please sign below, and give address and date.

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to fulfill this thesis. I am deeply indebted to my supervisors Professor Prabhakar Ragde and Professor Ming Li.

I thank Prabhakar for his supporting my study in the University of Waterloo, and encouraging me to find research topic that I am interested in. I also appreciate his many insightful comments that helped me to improve my papers, presentations and thesis. Ming has been my mentor in the exciting and growing field of Bioinformatics. I am most impressed his ability to seize new ideas, stimulating suggestions and unfailing encouragement that helped me in all the time of my research and writing of this thesis. I have also had the great fortune to benefit from the mentoring of Dr. Ying Xu and Dr. Dong Xu, both of the Oak Ridge National Lab, whose expertise in the field of protein structure prediction made it possible for me to build my own protein structure prediction program RAPTOR.

My special thanks go to Professors Daniel Brown, Ian Munro, Brendan McConkey and Philip E. Bourne for their generously agreeing to serve on my committee despite their busy schedules.

I am also grateful to several others. Professor Guohui Lin first introduced me to research in Bioinformatics when he was affiliated with the University of Waterloo; Professor Daniel Brown helped me very much in improving the writing of this thesis, and shared with me his skill in giving presentations, which, I believe, will benefit me throughout my future career; Tomáš Vinař has helped me numerous times in maintaining a highly dependable computing system; Robyn Lander has provided me kind help in using the supercomputer FLEXOR, without which RAPTOR could have achieved nothing in CAFASP3 (The Third Critical Assessment of Fully Automated Structure Prediction).

I would also like to thank Dr. Dongsup Kim who provided me with a huge amount of test data, Tina Li who helped to debug and test my computer program RAPTOR and prepare its manual, Gloria Rose who looked closely at the draft version of the thesis for English style and grammar, corrected both and offered suggestions for improvement, Dr. Daniel Fischer and Dr. Leszek Rychlewski who made it possible to automatically evaluate RAPTOR in a public manner, and several anonymous reviewers who provided helpful comments on RAPTOR and related papers.

I would like to give my special thanks to my parents and my wife Lily Liu, to whom I dedicate my thesis, for their endless love and spiritual support of my PhD study.

Abstract

If the primary sequence of a protein is given, what is its three-dimensional structure? This is one of the most important and difficult problems in molecular biology and has tremendous implication to proteomics. Over the last three decades, this issue has been intensely researched. Protein threading represents one of the most promising techniques. So far, there are many protein structure prediction computer programs based on protein threading; however, almost none incorporates the pairwise contact (interaction) potential explicitly in its energy function, although scientists believe that pairwise interactions are important for fold recognition targets. The underlying reason for ignoring the pairwise potential is that the protein threading problem is NP-hard (i.e., it is unlikely to have a polynomial-time algorithm), if the pairwise interactions are treated rigorously.

The key contribution of this dissertation is to show that there is actually a practically efficient algorithm for protein threading, even if the pairwise interactions are treated explicitly and rigorously. In this thesis, we propose a novel integer programming approach to solve the protein threading problem. The key element is a set of strong linear constraints to formulate the problem. In addition, we employ polyhedral combinatorics to analyze our formulation, and confirm that our formulation is well-described for the protein threading problem. Large scale experimental tests demonstrate that the optimal linear solutions of almost all the linear programs that formulate real threading instances solve to integrality, indicating that the protein threading problem is tractable in practice (i.e., it can be solved within polynomial time).

We have implemented the proposed algorithm combined with several other components as a protein structure prediction computer program RAPTOR (RApid Protein Threading predictOR). CAFASP3 (The Third Critical Assessment of Fully Automated Structure Prediction) evaluation, the latest worldwide protein structure prediction competition, ranks RAPTOR first among all the non-meta structure prediction programs. The latest LiveBench evaluation also shows that the performance of RAPTOR is superior compared to other structure prediction tools. These indicate that the integer programming approach is promising in further improving the prediction rate of the protein threading approach.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Over the last few decades, major advances in the field of molecular biology, combined with advances in genome sequencing technologies, have led to a large number of available genome sequences. Many computer tools have been developed to organize, analyze, and mine the exponentially growing genome sequence databases. What is next? For scientists in the post-genome era, the answer is proteins.

In her NIH (National Institute of Health) Record story [77], Alisa Zapp Machalek has described the importance of proteins in life as follows:

> "If genes are the recipes for life, then proteins are the culinary result—the very stuff of life. Proteins form our bodies and direct its systems. They digest our food, help us fight infections, control our body chemistry, and in general keep us—and every other living organism—running smoothly.
>
> But proteins that twist into the wrong shape, have missing parts, or don't make it to their job site can cause diseases that range from cystic fibrosis to cancer and Alzheimer's. "

Biochemists have established that a protein's spatial structure dominates its functions: the protein's role in health and diseases. Therefore, in attempting to understand a protein's

role in the body and to explore ways to control its actions, scientists often first determine its spatial structure. Usually, we use fold to refer to the type of a protein's spatial shape. The 3D structure of a protein is encoded in its gene which is transcribed into RNA and then translated into protein structures through exquisitely complex machinery. For example, the Human Genome Project has led to the identification of thirty to forty thousand genes in the human genome which may encode approximately 100,000 proteins as a result of alternative splicing [22]. Unfortunately, the experimental determination of protein structures is not as easy as genome sequencing. Currently, the 3D structure of a protein is solved using *x-ray crystallography* or *nuclear magnetic resonance spectroscopy (NMR)*. Both techniques are costly, time-consuming and difficult for high-throughput production, though technical advances have been achieved in recent years [120].

Conversely, relatively cheap and high-throughput methods exist to produce gene sequences and protein sequences. The question is whether or not we can predict the three-dimensional structure of a protein based on its sequence, one of the most important and long-standing problems in molecular biology. The difficulty of determining the 3D structure of proteins has led to an increasing gap between the huge number of protein sequences and the limited number of protein structures. As shown in Figure 1.1, the number of available protein structures in the PDB (Protein Data Bank) database is several orders of magnitude smaller than that of the available protein sequences based on the growth statistics of both databases [6, 5]. A high-throughput and affordable approach to generate protein structures is urgently needed in order to shorten the gap, and further understand biological systems. Such a method would revolutionize the proteomics industry [32].

1.2 Challenges

The overall strategy of the NIH Structural Genomics Initiatives, launched in 1999, is to solve protein structures using experimental techniques such as x-ray crystallography or NMR for only a small fraction of the proteins and to employ computational techniques to model the structures for the other proteins. The premise here is that although there may be millions of proteins in nature, the number of unique structural folds is probably two to three (or even more) orders of magnitude smaller [87]. Figure 1.2 depicts the

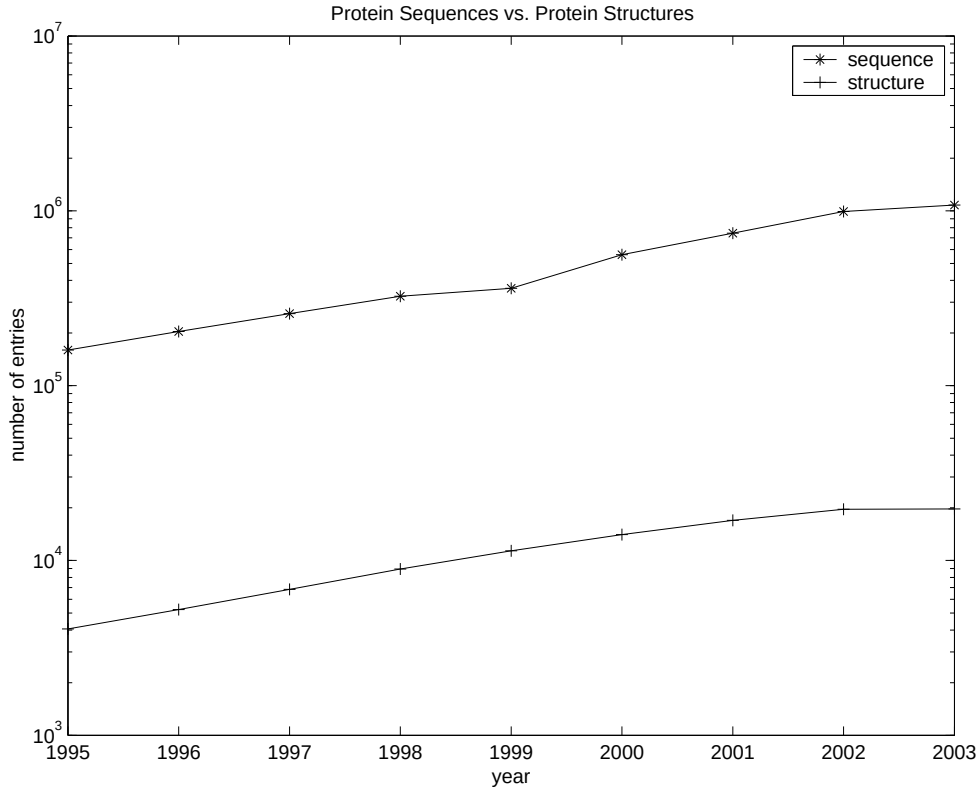

Figure 1.1: Growth rate of protein sequences and PDB in terms of the number of entries. The number of protein sequences is the size of the union of the nonredundant protein database and the regular releases of the following databases: SWISS PROT, PIR, Genpept and PDB.

proportion of new folds deposited to the available folds in the PDB database at each year from 1980 to 2001 [6]. We can see that on average, the proportion of new folds is no more than twenty percent. Consequently, by strategically selecting proteins with unique structural folds for experimental solutions, we can put the vast majority of other proteins "within modelling distance" of proteins whose structure is already known. Model-based structure prediction techniques can play a significant role in helping to achieve the goal of the Structural Genomics Initiatives. Protein threading represents one of the most promising such techniques.

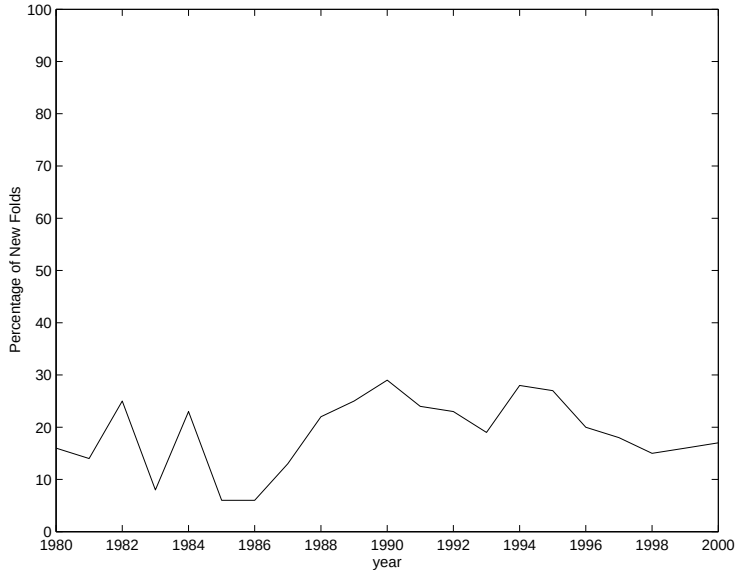

Figure 1.2: Proportion of new folds deposited in a given year, as a percentage of all the available folds in the PDB database.

As we mentioned before, the number of unique folds in nature is fairly small; therefore, we can construct a structural template database of several thousand entries from the PDB database by excluding those with highly similar sequences and structures. Each unique fold is represented by one or several templates. *Protein threading makes a structural prediction for the target sequence through finding the optimal alignment between the target sequence and each structural template in the template database and choosing the best-fit template as the basis on which to build structural model for the target.* If we can have a criterion (i.e., a scoring function) to quantify the quality of an alignment between the target sequence and the template, and an efficient algorithm to find the optimal alignment, then we can make a structure prediction for each target sequence within an affordable amount of time.

The criteria to evaluate the quality of the alignments between the sequence and the template include local conformation consistency and spatial conformation consistency. Often, the spatial conformation consistency is modeled as pairwise contacts, that is, if two amino acids in the template are spatially nearby (i.e., have a pairwise contact), then the two

amino acids in the sequence aligned to them should have strong pairwise contact potential (i.e., the probability of two amino acids being in two spatial nearby positions). So far, there are many available protein structure prediction computer programs [102, 60, 62, 52, 34] based on protein threading. However, almost none of these programs incorporates the pairwise contact potential into its energy functions, although biochemists believe that pairwise contact potential is important for protein folding. The underlying reason for ignoring the pairwise potential is that if the pairwise contact potential is considered in the scoring function, and variable gaps are allowed in the alignment between the sequence and the template, then the protein threading problem is NP-hard [67, 1], that is, computationally intractable. So far, there is no efficient algorithm which can treat the pairwise contact potential rigorously. This raises one question to be addressed in this thesis: Is there an algorithm which can treat the pairwise interactions rigorously and efficiently?

1.3 Goals

In light of the preceding discussion, we seek a protein threading algorithm capable of treating the pairwise contact potential rigorously and efficiently. Such an algorithm should have the following desired properties:

1. Effectiveness: The pairwise contact potential should be treated rigorously in searching for the optimal alignment between the sequence and the template. Both the alignment accuracy and the fold recognition rate should be comparable to state-of-the-art structure prediction programs. Such an algorithm should generate a much higher prediction rate and better alignment accuracy than those computer programs which downplay the pairwise contact potential in searching for the optimal sequence-template alignment.

2. Efficiency: Such a threading algorithm should run fast. Of course, it is difficult to outperform those algorithms that do not deal with the pairwise contact potential rigorously. But the new algorithm should terminate within a reasonable time. Fast genome sequencing demands the matching capability of fast structure prediction, which requires highly efficient prediction algorithms.

This thesis reports on the development of a very efficient and effective algorithm and computer program to do the protein threading and validates its performance by several experiments.

1.4 Contributions and Results

The main contribution of this thesis is to propose an (integer) linear programming (LP) approach to the protein threading problem (which is NP-hard), and to outline the design and implementation of a rapid protein structure prediction system RAPTOR (RApid Protein Threading predictOR). We show that, in practice, almost all instances of the intractable protein threading problem that arise in nature can be solved by linear programming. Linear programs have been the subject of active research for several decades. Moreover, they can be solved within polynomial time [21, 98]. Current linear program software packages such as IBM OSL can solve a linear program with a million of variables and millions of nonzero elements in the constraint matrix very fast [122]. Hence, we can solve the protein threading problem very efficiently in practice by the linear programming approach. Also, we show that RAPTOR can obtain a superior protein structure prediction rate for the hard target sequences that have no homologous proteins with known structures. The following paragraphs present the main components in our solution:

- **Integer program formulation to protein threading problem**: In this thesis, we propose three different integer programs, based on the pairwise contact map graph of protein 3D structures, to formulate the protein threading problem. We theoretically analyze the "tightness" of these three formulations and finally choose the best formulation as the default kernel in RAPTOR.

- **Theoretical analysis of the integer program formulation**: We solve the integer program by the canonical branch-and-cut method. We investigate if two types of well-known cuts can be used to strengthen our formulation. The surprising result is that these cuts are already implied by our formulation, which indicates that our LP formulation is well described for the threading problem due to the special structure of this problem. In addition, we prove that for a special case, our LP formulation

guarantees generation of the integral solutions directly without any branching.

- **Graph reduction technique**: There are some types of "partial regularity" in most of the contact map graphs of the protein templates, and the divide-and-conquer method proposed in PROSPECT-I [132] can very efficiently deal with those templates with a regular contact map graph. Based on this observation, we incorporate the divide-and-conquer method into our linear programming framework through a graph reduction technique. The objective of the graph reduction operation is to minimize the number of variables, the number of constraints, and the number of non-zero elements in the constraint matrix, which are key to the computational time of solving linear programs by both the Simplex method [21, 98] and interior point method [112]. In our LP formulation, the number of variables, constraints and non-zero elements in the constraint matrix are proportional to the number of edges in the contact graph. After the graph reduction, some subsegments of the template are reduced to a single edge. As a result, the original contact map graph is reduced to a new one with fewer vertices and edges. Experiments show that in most of cases, the speed of RAPTOR can be increased by approximately 30% in threading a long sequence to the whole template database. At the same time, the quality of protein threading is not impacted by the graph reduction technique because the scoring function is still globally optimized.

- **The Support Vector Machines (SVM) approach for fold recognition**: After threading the target sequence to each template in the database, we need to sift out the most probable templates for the structural model building. In contrast to the employment of the traditional z-score to recognize the correct fold from the template database [80], we adopt the SVM method to perform the fold recognition. The SVM approach is better than the traditional z-score in the following two aspects: First, the calculation of the SVM classification is inexpensive after training the SVM model. For each threading pair, the SVM approach needs to calculate only one vector inner product to do the prediction. The calculation of the traditional z-score is often time-consuming; after each threading, several thousands of random shufflings of the sequence must be carried out, in order to calculate the z-score for one threading pair.

Second, the SVM approach is more effective in choosing the correct templates than the traditional z-score.

1.5 Organization

The rest of this dissertation is organized as follows. The first part of Chapter 2 introduces several basic units of protein structures and the physico-chemical factors in determining the spatial shape of proteins. The second part of Chapter 2 surveys state-of-the-art protein structure prediction methods and introduces three types of prediction methods: *ab initio* folding, homology modelling, and fold recognition (protein threading).

Chapter 3 introduces some background material concerning protein threading such as using the Bayesian objective function to measure the alignment quality between the sequences and the templates, the computational complexity of the protein threading problem, and some existing algorithms to deal with the pairwise contact potential.

In Chapter 4, we present our linear program formulation of the protein threading problem. We propose three different sets of linear constraints to formulate this problem, analyze their tightnesses and compare them theoretically.

Chapter 5 theoretically analyzes the best integer program formulation. We analyze two well-known classes of cuts to see if extra constraints are needed to strengthen our formulation. Then experimental results on three large-scale real data sets are used to verify the theoretical results.

Chapter 6 describes how to exploit the partial regularity of protein contact map graphs. The original contact graph is reduced into a new one with fewer vertices and edges. The number of vertices and edges in the contact graph is closely related to the size of our LP formulation which, in turn, determines the needed computational time. We further incorporate the divide-and-conquer method proposed by Xu et al. [132] into the linear programming framework through graph reduction technique. We also discuss the generalization of our linear program formulation and the graph reduction technique in this chapter.

Chapter 7 details the implementation of our protein structure prediction computer program RAPTOR. We describes two essential components of our protein threading program:

the scoring system (i.e., the scoring function) and weight factor training. In this chapter, we also introduce the SVM approach to perform fold recognition, that is, to choose the correct template from the template database based on the optimal alignments between the target sequence and the templates.

Chapter 8 reports some experimental results on the structure prediction capability of our computer program RAPTOR. In this chapter, we present three kinds of tests: Large scale benchmark tests, the latest CAFASP3 evaluation, and the LiveBench-6 test.

Finally, Chapter 9 concludes the dissertation, discusses the strengths, extensions, and limitations of our work, and ends with directions for future work.

Chapter 2

Background

In an aqueous environment, a protein chain usually folds into a specific shape in order to function. The conformation of a protein can be parsed into several different levels ranging from the local structure to the overall spatial structure. There are several key factors (inter-atomic forces) that play instrumental roles in the formation of protein shapes. Almost all protein structure prediction programs model some or all of these factors in their scoring functions. This chapter provides an overview of the architecture of protein structures, the factors determining the overall spatial structure of proteins, and state-of-the-art protein structure prediction methods. We introduce protein structure in Section 2.1, then discuss several important physico-chemical factors in determining protein shapes in Section 2.2. Sections 2.3 to 2.5 contain a summary of three classical categories of protein structure prediction methods. Finally, in Section 2.7, an emerging protein structure prediction method, the consensus method, is summarized. This method predicts structures by combining the results of other methods.

2.1 Introduction to Protein Structure

Proteins are large, complex molecules consisting of amino acids, the basic building blocks of proteins. The spatial conformation of a protein is dominated by the order of the amino acids contained in it, and their side chain chemical properties. Protein spatial conformations can be described at four different levels: primary sequence, secondary structure, tertiary

structure, and quaternary structure. In this section, different levels of protein structures are briefly discussed [8].

2.1.1 Amino Acids

As shown in Figure 2.1, an amino acid consists of an amino group (-NH2) which is also called N-terminal, an α-carbon atom in its center, a hydrogen atom (-H), a carboxyl group (-COOH), and a side chain R group. Each amino acid is represented by one capital alphabetical letter. In nature, there are mainly 20 different types of amino acids that differ only in their R groups. These vary not only in their structure and size but also in their physico-chemical properties.[1] The structural complexity of the R group ranges from a simple hydrogen to a complex aromatic ring. According to the properties of their side chains, amino acids can be classified into four groups: hydrophobic, hydrophilic, positively charged and negatively charged. The hydrophobic amino acids tend to stay in the interior of the proteins, avoiding the outside aqueous solutions, whereas the hydrophilic ones are more likely to remain in the exterior of the proteins, interacting with the surrounding water molecules, and thus, stabilizing the shape of the proteins. Two oppositely charged amino acids can form a salt bridge. These interactions between amino acids influence the shape of the protein.

2.1.2 Four Levels of Protein Structures

Two amino acids are connected to form a peptide by a chemical reaction in which a water molecule is removed and the C in the carboxyl group (-COOH) of one amino acid is linked directly to the N in the amino group (-NH2) of the other amino acid. Figure 2.2 illustrates an example of a peptide. Multiple amino acids can be connected sequentially in such a way to form a polypeptide. The amino acids contained in a polypeptide are called residues. Figure 2.3 shows an example segment of a polypeptide.

The primary sequence of a protein describes the linear order of amino acids contained in it. The primary sequence of a protein starts from the NH2-group (N terminal), and ends at the carboxyl group (C terminal). Although the primary sequence does not explicitly express

[1] There are also several other types of amino acids in nature, but they are relatively scarce.

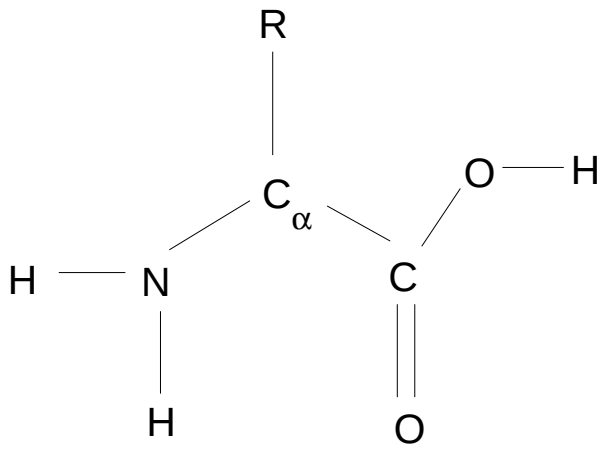

Figure 2.1: Structure of an amino acid [93].

any spatial structure information about the protein molecule, the spatial conformation of proteins is determined by their primary sequences [4]. The folding of a small fraction of most newly synthesized proteins in the cell needs the assistance of some type of molecular chaperone because of the folding environment in the cell. Chaperones act like catalysts in the sense that they temporarily interact with their substrate proteins but are not present in the final folded product [33]. Thus, it is still fair to say that the three-dimensional structure of a protein is determined by its primary sequence.

Proteins fold up to complex shapes due to the bonds formed between side chains. Not only are there strong bonds among two residues which are nearby along the primary sequence, but there can be strong bonds between two residues which are far away from each other. The former ones are called local interactions or short-range interactions; and the latter ones are called non-local interactions or long-range interactions. Interactions between two residues are also called pairwise contacts.

A segment of protein primary sequence can fold into a secondary structure because of the short-range interactions. Two common types of secondary structures are the right-handed α-helix (see Figure 2.4 for an example), and the β-sheet (see Figure 2.1.2) which can be connected into a larger tertiary (global) structure by various loops. There are strong hydrogen bonds among the residues within one secondary structure. In contrast, the bonds

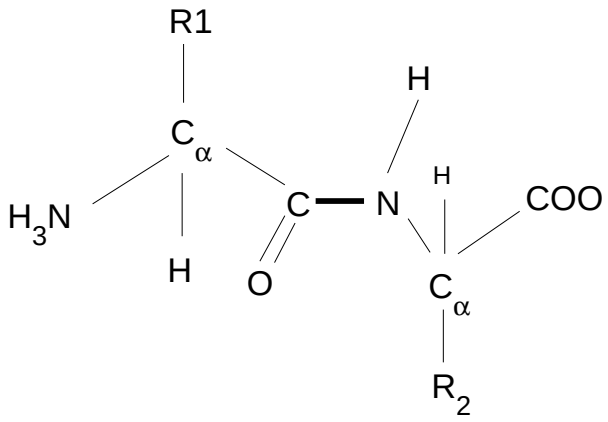

Figure 2.2: Peptide, the amide bond is represented as a thick line.
taken from [117]

among the residues within one loop are weak.

An α-helix is formed by strong hydrogen bonds between two residues which are four positions apart in the primary sequence. An α-helix looks like a spiral in which the α-carbon atoms lie in the inner part, whereas the side chains are on the outside. Often, an α-helix is represented as a cylinder or a ribbon forming a spiral in the structure drawing. Figure 2.4 shows an example of an α-helix with three different representations.

A β-sheet is formed by a parallel or, more commonly, an anti-parallel arrangement of individual β-strands which are tied together by hydrogen bonds. The hydrogen bonds are formed by the amino group and the carboxyl group of different strands, as opposed to the same strand in the α-helix. A directional arrow is the simplest representation of a β-strand in the structure diagram. Figure 2.1.2 illustrates a parallel and an anti-parallel beta sheet with two different representations.

If a combination of several secondary structure units occurs frequently, then this combination is called a super secondary structure. A super secondary structure can be found in various kinds of proteins that unnecessarily have the same fold. Typical examples are the helix-turn-helix, the beta-alpha-beta, and the coiled coil (two α-helices twisted around each other) motifs. Super secondary structures can be used to predict protein three-dimensional structures. For example, in their new fold structure prediction program FRAGFOLD [54],

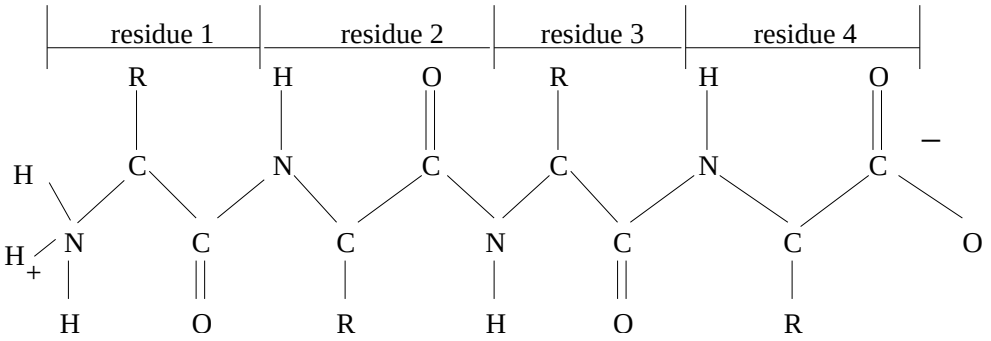

Figure 2.3: Segment of polypeptide
taken from [117]

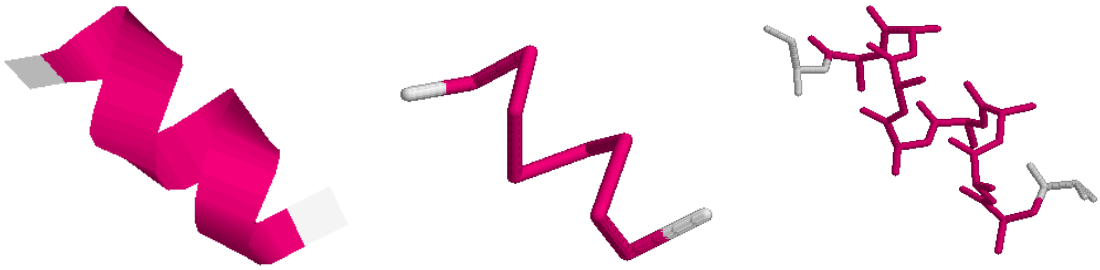

Figure 2.4: Example of α-helix with three different representations, drawn by RasMol.

Jones et al. predicted the three-dimensional structure of a protein by sampling from its all possible super secondary structures.

Due to the long-range interactions, all secondary structures in a protein can form a specific tertiary structure with the loops connecting one secondary structure to another. Figure 2.6 presents the 3D structure of an example protein chain 12asa[2], taken from the PDB database [6].

Frequently, we use folds to denote the type of tertiary structure of a protein. After a long period of evolution, the sequences of proteins are extremely varied, whereas the 3D structures are much more restricted [85, 47, 88, 90, 87] because a fraction of residue

[2]12as is the PDB ID and a is the chain label.

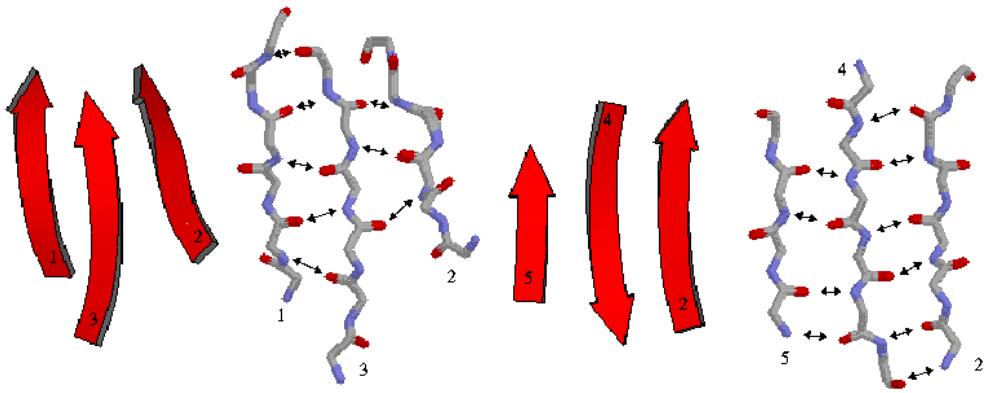

Figure 2.5: Example of parallel (left) and anti-parallel (right) beta sheets, each of which has two different representations, taken from http://broccoli.mfn.ki.se/pps_course_96/ .

exchanges does not affect the stability of structures. Therefore, one single fold can correspond to many protein sequences with very low number of identical residues. According to the work of Rost [92], if two proteins of length no less than 100 residues have the sequence identity of more than 35%, then these two proteins have very similar structures. In spite of a large number of different proteins in nature, there are many fewer (approximately 1000) different protein structural folds [87], which is the basis of the success of the protein threading method for protein structure prediction.

Many large globular proteins consist of several polypeptide chains which are kept together by various forces such as hydrogen bonds or disulfide bonds though each chain can fold independently. Such a protein is called a multiple-chain protein or a complex. For multiple-chain proteins, quaternary structure represents the spatial relationship among all the proteins' chains. The protein complex shown in Figure 2.7 consists of two chains.

A protein chain can be divided into several different domains, each of which can have its own structural fold and functions and believed to be able to fold independently [85, 88, 90]. The average length of one domain is approximately 140 residues [23, 119]. If the three-dimensional structure of a protein is known, it is relatively easy to determine the domain boundary of all the domains contained in this protein; an expert can determine

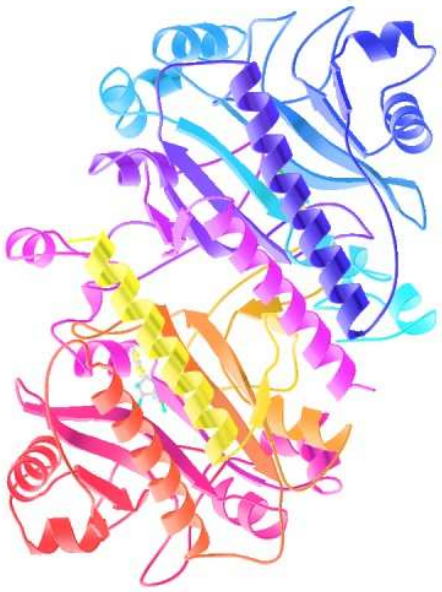

Figure 2.6: 3D structure of a protein chain 12asa, taken from PDB database.

it by observation. Some good computer programs [131, 43] can parse protein domains automatically if its three-dimensional structure is available, but it is not as easy to parse the domain boundary of a protein sequence without known structure. Some programs have been developed to parse the domain boundary of a protein sequence, e.g., ProDom [23, 100]. Knowing the accurate domain boundary of a protein sequence often helps to improve its structure prediction accuracy. The structure of a large protein with multiple domains is harder to be predicted by the protein threading method. Usually, there is no template in the database for such a large protein. It is possible that each domain of this large protein can be aligned very well by a template, but only one template can be ranked first for the whole protein. In order to predict the structure of the whole large protein, a better way is to split the large protein into several domains in advance and then predict the structure of each domain. In Chapter 8, we will discuss a failure example of the prediction of a multiple-domain protein.

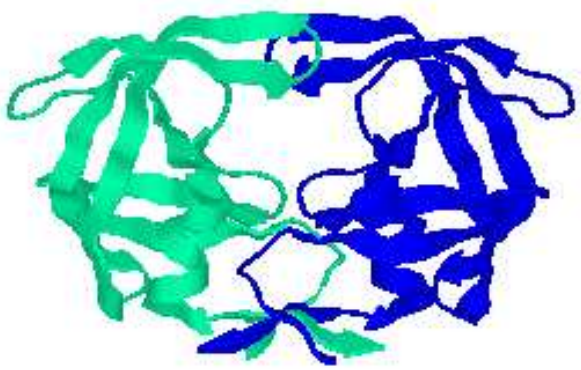

Figure 2.7: Example of quaternary structure.

2.2 Factors Determining Protein Structures

In order to successfully predict the structure of a protein based on its primary sequence, we need to understand the underlying principles concerning the formation of a protein's shape. As already mentioned, proteins fold up because of the different properties of side chains. It is these different properties of side chains that lead to four major inter-atomic forces stabilizing the shape of a protein: hydrogen bond, hydrophobic effect, Van der Waal force and electrostatic force [11, 73].

Hydrogen bonds are the most important forces in stabilizing secondary structures such as α-helix and beta sheets. The strength of hydrogen bonds weakens rapidly with alternations in the bond angle. Van der Waal forces are formed by the attractions between an atom's nucleus and its neighbour's electrons. Van der Waal forces are isotropic and weak, but a number of such interactions can help to stabilize the central hydrophobic core of proteins. Electrostatic forces result from the oppositely charged side chains. They are quite strong, and are key factors in stabilizing secondary and tertiary structures. Sometimes, they also function as the inhibitor of a certain fold.

Most of protein threading-based structure prediction methods do not model these forces in detail. Instead, they employ a knowledge-based method to estimate the contact potential between any two amino acids.

2.3 Protein Structure Predictions

The protein structure prediction problem is to predict the tertiary structure of a protein from its amino acid sequence. In attempt to fulfill this overall goal, two related subproblems, protein secondary structure prediction and residue-residue contact prediction, are also being actively researched. There is an increasing gap between the number of existing protein sequences and that of the known protein structures due to various genome sequencing projects around the world. In an attempt to identify protein structures quickly, the researchers are trying to predict protein tertiary structures from their sequences by using existing biological knowledge and computational methods. The primary sequence of a protein contains the necessary information for determining its conformational arrangement, and thus it is feasible to predict the tertiary structure of a protein based on its sequence [4].

Many protein structure prediction methods are based on a basic thermodynamic hypothesis: proteins tend to fold into a global minimum free energy state. Consequently, researchers predict protein structures in two steps: first, design a scoring function to reflect the relationships among the amino acids in the native states, and second, design or employ certain algorithms to optimize the scoring function. The result is the three-dimensional structure of a protein. The scoring function can be constructed from the physico-chemical principles of protein folding, and reflects the real energy of proteins in a native state. Alternatively, the scoring function can be a knowledge-based potential function, measuring the probability distribution of the possible conformational arrangements of a protein sequence. Traditionally, the scoring function is also called "energy function" even if the scoring function does not reflect the real energy of proteins.

During the last three decades, many protein structure prediction methods have been proposed, and in the past ten years, many structure prediction computer programs have been developed to ease or speed up manual predictions. The methods can be grouped into three categories: homology modelling, fold recognition (protein threading) and *ab initio* folding (new fold method), according to their most suitable target sequences. The first two are template-based, and the third one builds protein structures without referring to any structural template.

Homology modelling is for those target sequences obviously having homologues with

a known three-dimensional structure. This kind of target sequences are called homology modelling targets. Homology modelling builds the tertiary structure of a target sequence by comparing the target sequence to all of its homologous sequences, recognizing the most conserved segments through multiple alignments, copying coordinates for these conserved segments from one homolog with a known structure, and finally, refining the whole structure through the energy minimization technique. Fold recognition or protein threading is suitable for those target sequences (called fold recognition targets) which have no homologous templates but do have the same fold as some templates. The three dimensional structure of a target sequence is built by placing its amino acids one-by-one and sequentially into different positions of the template which has the same fold as the target sequence.

Ab initio folding is the preferred methodology to apply to those targets that could not be predicted by the two former methods, that is, the targets that do not have the same fold as some templates or do not have the homologous proteins with a known structure. This kind of targets are called new fold targets. *Ab initio* folding has also been called the "new fold" method in recent years, because it is appropriate for targets that do not have the same fold as any protein with a known structure, and the techniques used deviate dramatically from the traditional lattice model simulation. *Ab initio* folding depends on a scoring function which can accurately describe the native state of proteins and on an efficient algorithm to optimize the scoring function.

Besides the traditional three structure prediction methods, a class of computer programs have been recently developed to combine the outcomes of several different prediction programs by certain consensus methods, such as 3DSX, developed by Daniel Fischer [35], 3D-Jury by Lesek Rychlewski [38], and PMODX by Janu Bujnicki [63]. Because of their distinct differences from the three previous categories, we classify these programs into a new category called the consensus method. In the following sections, we will summarize these methods, and study some representatives of each method.

2.4 Homology Modelling

2.4.1 Introduction

The homology modelling method constructs the three-dimensional structure for a target sequence by using the homologous proteins of the target. Homology does not necessarily mean sequence similarity or structure similarity, although homologous proteins have a similar sequence and structure to some degree. Several proteins are regarded as homologous because they are evolved from the same ancestor. Homologous is a qualitative rather than a quantitative description of the relationship between two proteins. Therefore, we can say two proteins are homologous or not homologous, but not that they are partially homologous. A underlying premise for homology modelling is that if a set of proteins are homologous, then their three-dimensional structures are more conserved than their primary sequences [20, 89, 94]. Homology modelling is extensively researched and the most reliable model structure building method on the basis of the known three dimensional structure [108, 45, 72, 71, 44, 18, 82]. The homology modelling method for protein structure prediction consists of the following procedures:

- **Homologue Selection**: Identify one or several homologous proteins (parent templates) from the structure database and determine the extent of their sequence similarity with the target sequence. Some computer tools such as PSI-BLAST [3] can be used.

- **Sequence Alignment**: Build a multiple sequence alignment among the target sequence and the selected homologous sequences.

- **Core Determination**: Identify the most conserved segments and variable segments in the multiple sequence alignment. The most conserved segments are the cores of these homologous proteins, and generally lie in the secondary structures; the variable segments are treated as loops.

- **Core Modelling**: Predict the coordinates of the backbone atoms of the recognized cores of the target by copying them from those of the aligned backbone atoms of one

homologous protein. Use a rotamer library to generate the coordinates of the side chains of these cores [97, 70].

- **Loop Modelling**: If a loop in one homologous protein is judged as a good model for one loop in the sequence, then copy the coordinates from the homologous protein to the target sequence; otherwise, search the existing fragment database to find a best-fit fragment for the loop. The coordinates of the side chains can also be constructed by using a certain structure fragment database [56, 91, 107].

- **Refinement and Evaluation**: After generating the coordinates for the atoms of the target sequence, evaluate and refine the generated structure. A general method is to tune the coordinates of some atoms through minimizing certain energy functions. Some tools such as WHATIF [116] and PROCHECK [81, 64] can be also used to check the quality of the constructed model.

2.4.2 Sequence Alignment

Sequence alignment is the key step in protein structure prediction by the homology modelling method. Four key factors should be carefully considered in this step:

- The algorithm to be used for pairwise or multiple sequence alignment;
- The scoring system to be used to measure the sequence similarity;
- How to penalize alignment gaps; and
- Which alignment strategy to use, local alignment or global alignment?

For pairwise alignment, a dynamic programming algorithm such as Needleman-Wunsch [86] and Smith-Waterman [106] algorithm can fulfill the task efficiently regardless of whether the alignment is local or global. But if multiple parent templates are available, there is no known polynomial-time algorithm for multiple sequence alignments [49]. Consequently, heuristics must be used.

Usually, mutation matrices (or substitution matrices) are used for scoring the similarity among the sequences. Mutation matrices are generated from the observation of how often

one given amino acid is replaced by another one among the proteins for which their sequences can be aligned. PAM and BLOSUM are two examples of the most commonly used mutation matrices. PAM (Percentage of Acceptable point Mutations), based on the global multiple alignment, was developed by Dayoff et al. [26, 99, 25]. Each entry of the PAM matrix gives the probability of one amino acid mutating to another amino acid within the given evolutionary time. There are many PAM matrices corresponding to different evolutionary times. "PAM" is often followed by a number indicating that this PAM matrix corresponds to the given time. The smaller the number, the shorter the evolutionary time, and the more similar sequences the matrix comes from. BLOSUM (BLOck SUbstitute Matrix) [46] was developed by Henikoff et al. and it is based on local multiple alignments. Henikoff constructed hierarchical clusters for the sequences according to the sequence identity, and then calculated the BLOSUM matrix for each cluster. Similarly, "BLOSUM" is often followed by a number showing that this BLOSUM matrix is constructed from the cluster within the given amino acid identity among all the sequences. In addition, there are other mutation matrices such as JO matrices [51], developed by Johnson and Overington according to the three-dimensional structure alignment. An affine gap function with an open gap penalty and gap extension penalty is used to penalize the alignment gap.

2.4.3 Study of Examples

The programs discussed in this section are just for the identification of the homologous templates and generation of sequence alignments. There are some specific computer programs such as MODELLER [96] and SWISS-MODEL [42] for the construction of protein structure models from sequence alignments among homologous proteins, though MODELLER and SWISS-MODEL can also be used to do homology modelling if the sequence and the template have high sequence similarity.

PDB-Blast [74] first employs PSI-BLAST [3] to generate a sequence profile for the target sequence, and then matches the sequence profile against the PDB structure library to find the best template. According to the CAFASP3 evaluation [37], PDB-Blast performs well for easy homology modelling targets, but it is not comparable with those fold recognition servers for hard homology modelling targets and fold recognition targets.

SAMT (Sequence Alignment and Modelling Tools) [59] is a software suite of the hidden

Markov model (HMM) method for protein sequence comparison. In each position of the HMM model, there are three states: insertion, delete, and matching. State transitions are allowed between two adjacent positions only. At each state, each of the 20 amino acids is output according to a certain probabilistic distribution derived by training on examples. The multiple sequence alignment among one family of proteins is used to determine the state transition probability and residue emission probability at each state. One HMM model must be constructed in advance for each family or each superfamily. The target sequence is matched against each HMM model to determine if it belongs to its representative family or superfamily. SAMT99 is suitable for easy homology modelling targets, but cannot handle fold recognition (FR) and new fold (NF) targets well. The latest version, SAMT02 contains more information such as solvent potential and predicted secondary structures into the HMM model. Therefore, it cannot be regarded as a solely sequence alignment based method any more.

2.5 Fold Recognition

2.5.1 Introduction

The fold recognition method, which is also called protein threading, is for those targets which have the same fold as proteins of known structures but do not have homologous proteins with known structure. Protein threading predicts protein structures by using statistical knowledge of the relationship between the structure and the sequence. The prediction is made by "threading" (i.e., placing, aligning) each amino acid contained in the target sequence to a position in the template structure, and evaluating how well the target fits the template. After the best-fit template is selected, the structural model of the sequence is built based on the alignment with the chosen template. The protein threading method is based on two basic observations. One is that the number of different folds in nature is fairly small (approximately 1000), and the other is that according to the statistics of PDB, 90% of the new structures submitted to PDB in the past three years have similar structural folds to the ones in PDB [6]. A general paradigm of protein threading consists of the following four steps:

- **The construction of a structure template database**: Select protein structures from the protein structure databases as structural templates. This generally involves selecting protein structures from databases such as FSSP [47], SCOP [85], or CATH [88, 90], after removing protein structures with high sequence similarities.

- **The design of the scoring function**: Design a good scoring function to measure the fitness between target sequences and templates based on the knowledge of the known relationships between the structures and the sequences. A good scoring function should contain mutation potential, environment fitness potential, pairwise potential, secondary structure compatibilities, and gap penalties. The quality of the energy function is closely related to the prediction accuracy, especially the alignment accuracy. The scoring function for protein threading is further detailed in Section 3.1 and 7.1.

- **Threading alignment**: Align the target sequence with each of the structure templates by optimizing the designed scoring function. This step is one of the major tasks of all threading-based structure prediction programs that take into account the pairwise contact potential; otherwise, a dynamic programming algorithm can fulfill it. This thesis is mainly dedicated to solving the optimal alignment problem derived from a scoring function considering pairwise contacts.

- **Threading prediction**: Select the threading alignment that is statistically most probable as the threading prediction. Then construct a structure model for the target by placing the backbone atoms of the target sequence at their aligned backbone positions of the selected structural template.

2.5.2 Study of Examples

PROSPECT has two versions with dramatic differences that have been developed by a research group at the Oak Ridge National Laboratory. The first version of PROSPECT [132] uses a divide-and-conquer method to treat the pairwise potential strictly in aligning the target sequences to the templates. But it runs very slowly if the templates have complex interaction topologies and the target sequences are long. The scoring function optimized

by the first version of PROSPECT contains mutation potential, environment fitness potential, distance-independent pairwise potential, secondary structure compatibilities and gap penalties.

The first version of PROSPECT performs very well in recognizing the fold recognition targets. PROSPECT-II [62] discards the strict treatment of pairwise interactions to speed up the search for the optimal alignment in order to fulfill the genome-wise structure prediction. The program aligns the target sequences to the templates by a dynamic programming algorithm regardless of pairwise contact potential. After aligning the sequence and the template, PROSPECT-II then calculates the distance-dependent pairwise score based on the existing alignment. Besides the non-pairwise z-score, the pairwise z-score is also calculated by randomly shuffling the sequence. A linearly combined z-score is calculated to select the best-fit templates. Just like any threading program based on dynamic programming, PROSPECT-II runs very fast, but it does not seem to work as well as the first version of PROSPECT in recognizing the fold recognition targets, according to the CAFASP3 evaluation result [37].

3D-PSSM [60] calculates three different alignments between a target sequence and a template by using different scoring functions and different alignment policies. The alignment with the highest standardized score is taken as the final one for this pair. In each alignment, the scoring function contains secondary structure information, solvent potential, and PSSM (Position Specific Scoring Matrix) information, as well as a gap penalty. 3D-PSSM employs two types of PSSMs: one-dimensional PSSM from the multiple sequence alignment of a family of proteins and three-dimensional PSSM from the multiple sequence alignment of a superfamily of proteins. The three different alignments are: the target sequence is aligned to the one-dimensional PSSM of the template; the target sequence is aligned to the three-dimensional PSSM of the template; and the template sequence is reversely aligned to the one-dimensional PSSM of the target sequence. Since all alignments are involved with only sequence to profile alignment, a dynamic programming algorithm can be used to search for the optimal alignment. 3D-PSSM uses PSI-BLAST [3] to generate one-dimensional PSSMs for target sequences or templates and then constructs the three-dimensional PSSM for each template by combining all one-dimensional PSSMs of the proteins which are in the same superfamily as the template.

FUGUE [102] aims at the recognition of distant homologs. It aligns a sequence to a template by comparing the PSI-BLAST profile of the sequence with the structure profile of the template. At each template position, the structure profile is an environment substitution table constructed from the multiple structure alignment among the homologous proteins. Also, a position-dependent gap penalty is used in the scoring function. At each position, the gap penalty is dependent on the solvent accessibility at this position, and its position relative to the secondary structure elements.

Similar to 3D-PSSM [60], SHGU [34] makes use of five different threading methods to match the target sequence against the template library and then rank the templates by combining the outcomes of these five methods. The five methods are: (1) the alignment between the target sequence and the template sequence; (2) the alignment between the multiple sequence alignment profile of the target sequence and the template sequence; (3) the alignment between the PSI-BLAST profile of the target sequences and the template sequence; (4) the alignment between the target sequence and the PSI-BLAST profile of the template; and (5) the alignment between the multiple sequence alignment profile of the target sequence and the PSI-BLAST profile of the template. The final z-score is the average of the five individual z-scores calculated from the five different alignments. SHGU performs very well for all the targets in both the latest CAFASP3 evaluation [37] and LiveBench-6 [95].

2.6 *Ab Initio* Folding

2.6.1 Introduction

The *ab initio* folding method builds the structural model of a protein sequence directly from the primary sequence alone, without resorting to any parent structural template. Scoring functions employed for the *ab initio* folding methods includes both the traditional one which models the atomic force fields [12, 118], and knowledge-based ones developed in recent years [104]. This difference is due to the two different approaches that are taken by the researchers working on *ab initio* folding. One is to start from simulating the folding pathway of a protein and finally building the structural model of the sequence, whereas

the other approach directly predicts the most probable structure of a protein sequence by searching the entire conformation space of the sequence.

If we know the physical environment where the proteins fold, then, in principle, we can simulate the folding pathway by algorithmic implementations of the physical laws [79, 110, 28] for atomic interactions. The successful stories of this approach are scarce for two reasons. First, researchers do not yet have a complete and clear understanding of the underlying mechanisms of protein folding. Second, this approach involves modelling the interactions of all the atoms in a system of many thousands of atoms for dozens of microseconds, at a time step of femtoseconds. Even the fastest available supercomputer is inadequate for the large CPU time requirement for a medium-sized protein sequence [28]. Some approximation methods such as Monte Carlo simulation and vibrational analysis have been employed to speed up the simulation. Duan and Kollman have successfully simulated the folding of a protein villin headpiece of 36 amino acids for just one microsecond with 256 Cray processors running for two months [28]. The total number of time steps is 500 million, with a time step of 2 femtoseconds.

In recent years, direct conformation space search methods have been more successful than the folding pathway simulation. The computer program, Rosetta [104], is an outstanding representative. However, exhaustive conformation space search is still formidable given current computing speeds, due to a high number of candidate conformation arrangements. In an attempt to reduce the search space, researchers have adopted several strategies such as simplifying the representation of the proteins or restraining the conformation space. For example, an amino acid is abstracted as one or two points, although the coordinates of all the atoms contained in it must be determined. Some algorithms also add some restraints such as sparse NMR data and the predicted residue-residue contacts to limit the geometry of the protein structures and, as such, scale down the search space.

Whatever the strategy is, some optimization technique must be used to minimize the energy function in order to find the most probable model. The commonly used techniques include Monte Carlo sampling, simulated annealing, and genetic algorithms [104, 54, 105]. But these algorithms may not converge to the global optimum within polynomial time or ever. Although *ab initio* prediction has been extensively researched for several decades, the success rate is still discouraging. In fact, the structures predicted by the *ab initio*

folding method are less useful than those by the homology modelling method and the fold recognition method, due to their very low accuracies.

2.6.2 Study of Examples

Rosetta [104] is the state-of-the-art *ab initio* folding method. Rosetta constructs a structure fragment library from the PDB database by a structure clustering technique. Each entry in the fragment library has a length ranging from three to nine residues. The mapping from one sequence segment to multiple small structure fragments is also built, according to the local sequence-structure correlation [17]. To overcome the extreme computational demands, Rosetta cuts the whole target sequence into dozens of (overlapped or non-overlapped) segments of length from 3 to 9 residues. The spatial structure for each segment is sampled from the fragment library according to the sequence-structure relationship. A Monte Carlo sampling technique with a scoring function favoring nonlocal interactions, paired β-strands and buried hydrophobic residues is employed to sample a whole 3D structure for the target sequence. In order to speed up the sampling procedure, some filters eliminating non-protein-like conformations are adopted as follows: (1) conformations with poorly formed β-sheets are eliminated; (2) for long target sequences, conformations with higher contact orders are preferred; and (3) the conformations with poorly packed interiors are eliminated. For each target sequence, thousands of independent simulations are carried out. Besides generating possible conformations for the target sequence itself, Rosetta independently samples conformations for the sequences in the same family as the target. The final conformations of all simulations are clustered and the centers of several of the largest clusters are chosen as the most probable models.

In contrast to the small sequence segments used by Rosetta, FRAGFOLD [54] predicts structure by assembling some supersecondary structure fragments. Each supersecondary structure fragment is a candidate structure for a target subsequence composed of several segments of the sequence. All structural fragments are taken from highly resolved protein structures. FRAGFOLD uses simulated annealing to explore the possible combinations of the supersecondary structure fragments. In addition to the solvation potential and pairwise potential in the scoring function, FRAGFOLD uses two extra items to measure the compactness of the folds and avoid steric clash, respectively. For each target sequence,

20 separate simulations are carried out simultaneously, and the final 20 conformations are clustered. A representative of each cluster becomes the final model.

Skolnick et al's method differs from Rosetta and FRAGFOLD [105]. First, it predicts the residue-residue contacts by threading the target sequence to their own template library. Some predicted contacts are randomly chosen as the extra distance constraints to the folding simulation. After the Monte Carlo technique is used to build 20 to 50 initial lattice models, off-lattice modelling is done. The last step of fold selection is similar to the previous two methods; a certain clustering technique is employed to select the representative predictions.

2.7 Consensus Method

2.7.1 Introduction

Protein structure prediction protocols that combine the outcomes of multiple structure prediction programs use the "consensus method". The programs employing the consensus method are called meta-servers. In recent years, there have been an increasing number of meta-servers because they can produce a more sensitive and specific prediction than individual (non-meta) servers. The first generation of meta-servers such as, Pcons [76], the first reported meta-server, usually chose one prediction from their inputs as the best prediction by using a certain voting scheme or machine learning method. They did not fully exploit the information provided by the individual servers. Contrary to their old counterparts, the latest meta servers such as 3DSX [35] and PMODX [63] can now employ fragment assembly techniques to construct a new consensus model from their inputs (*i.e.*, the constructed model is not necessarily a copy of one of the inputs any more), which is an impressive and promising advance.

2.7.2 Study of Examples

Pcons [76] is the first reported consensus fold recognition predictor. The inputs of Pcons are the outcomes of several publically available fold recognition servers such as 3D-PSSM [60], FUGUE [102] and GenThreader [52]. Pcons makes a prediction by exploiting two

types of information provided by the individual servers: the z-score and the similarity of any two models produced by two different servers. The similarity of the two models is measured in two different ways: one is the structural similarity of the two predicted structures and the other is the structural similarity of the two parental templates. A two-layer neural network is trained to select the best models. In the primary layer, there is a sub-neural-network for each server to predict the quality of a model based on the z-score and the number of other similar models. In the second layer, one jury neural network is used to combine the outputs of the primary neural networks. It was reported that Pcons performed at least as well as the best individual server [76] in sensitivity on all CASP4 targets. However, the developers claimed that the improved Pcons was more sensitive on all LiveBench2 targets [15] than any individual server [30]. Just like other meta servers, Pcons can greatly improve the prediction specificity [76, 30].

A newer program, 3D-SHOTGUN [35], constructs a consensus model for a target sequence by making use of a certain fragment assembly technique. First, it calculates the structural similarity of all the pairs of input models. For each model, 3D-SHOTGUN chooses those models which are similar to this model and then reconstructs the coordinates for each position of this model. The coordinate of each position is the most frequent occurring coordinate among all the chosen models. After generating the coordinates for each model, a confidence score is assigned to each model. All generated models are ranked according to their confidence score. There are several variants of 3D-SHOTGUN in terms of the individual servers used. However, it is not true that as the number of individual servers used grows, so does the quality of results that 3D-SHOTGUN generates. According to the CAFASP3 evaluation [37], 3DS5, which uses five individual servers is the most sensitive and specific among all participant servers. 3DS3, a meta server constructed from 3 individual servers, also performs very well. 3DSN, making use of the outcomes from almost all individual servers, performs worse than 3DS5 and 3DS3. The possible reason is that some noises are introduced by those servers with poor performances.

Chapter 3

A Survey of Protein Threading

Protein threading is proposed for those target sequences that have the same fold as some proteins with known three-dimensional structures but do not have homologous proteins with known structures. Protein threading makes a structure prediction through placing (aligning) the residues of the target sequence sequentially to the positions in the template to see if the target can have the same fold as the template or not. Gaps are allowed in the alignment to some extent. Not all sequence residues are aligned to a template position and not all template positions are aligned by a sequence residue. For the successful application of protein threading in structure prediction, two kernel problems should be addressed. One is the design of the energy function to measure the quality of the alignments between the target sequences and the templates and the other is an efficient alignment algorithm to search for the optimal alignment between the target sequence and the template according to the given energy function.

In this chapter and the following ones, we only consider the energy function which contains pairwise contact potential. This results from the fact that the dynamic programming algorithm can solve the optimal alignment problem very efficiently if pairwise contact potential is not considered [60, 102]. In Section 3.1, we describe a commonly used Bayesian-based energy function. Sections 3.2 and 3.3 survey the computational complexity results and existing algorithms regarding the threading model in which the pairwise contact potential is taken into account.

3.1 Energy Function

The energy function should be able to quantitatively measure the quality of a given alignment between the target sequences and the templates. Generally, the energy function contains sequence mutation item, environment fitness item and structure consistency item [132, 14, 128, 127, 126, 60, 102, 52]. Sequence mutation information refers to the mutation potential between a sequence residue and a template residue. The environment fitness information measures how well one sequence residue has been placed in the solvent environment at one given template position. The structure consistency information consists of two components. One is the local structure consistency (*i.e.*, the secondary structure compatibility) and the other is the overall structure consistency (*i.e.*, the pairwise contact consistency or multiple-body contact consistency). For each type of information contained in the energy function, there is a weight factor to control its importance in the measurement of the alignment quality. The weight factors should be adjusted for different types of target sequences. For example, if only the sequence mutation information is considered, then the threading method is, de facto, a sequence alignment method which is used for the identification of homologous templates. If only the structure consistency information is considered, then the threading method is suitable for recognizing only those non-homologous templates that have the same fold as the target. If the three types of information are used together in the energy function, then the threading method may be able to recognize additional templates, depending on the weight factors setting.

Formally, we can use the Bayesian method to deduce the objective energy function [65]. Let $P(T|S)$ denote the probability of the target sequence S being in the same fold as the template T. Let $A = (A(1), A(2), ..., A(n))$ denote an alignment between the target sequence and the template where the residue in the sequence position j is aligned to the template position $A(j)$. Let $P(T|S, A)$ denote the probability of the sequence being aligned to the template according to the given alignment A. Then we have

$$P(T|S) = \max_A P(T|S, A) \tag{3.1}$$

Applying Bayesian rules, we have:

$$P(T|S, A) = \frac{P(T, S|A)}{P(S)}$$

$$= \frac{P(S|T,A)P(T)}{P(S)} \tag{3.2}$$

If we assume that $P(T)$ is a uniform distribution, given a specific target, then based on Equation 3.1 and 3.2, we have

$$P(T|S) \propto \max_{A} P(S|T,A) \tag{3.3}$$

Assume the target sequence S be $a_1, a_2, ..., a_n$ and the template sequence be $T = t_1, t_2, ..., t_m$. $P(S|T,A)$ can be expanded as follows:

$$\begin{aligned} P(S|T,A) &= P(a_1, a_2, ..., a_n | t_1, t_2, ..., t_m, A) \\ &= \prod_i P(a_i|t_{A(i)}) \prod_{i<j} \frac{P(a_i, a_j|t_{A(i)}, t_{A(j)})}{P(a_i|t_{A(i)})P(a_j|t_{A(j)})} \\ &\times \prod_{i<j<k} \frac{P(a_i, a_j, a_k|t_{A(i)}, t_{A(j)}, t_{A(k)})P(a_i|t_{A(i)})P(a_j|t_{A(j)})P(a_k|t_{A(k)})}{P(a_i, a_j|t_{A(i)}, t_{A(j)})P(a_i, a_k|t_{A(i)}, t_{A(k)})P(a_j, a_k|t_{A(j)}, t_{A(k)})} \cdots \end{aligned} \tag{3.4}$$

The first item of the right hand side (RHS) of Equation 3.4 is the probability of one particular amino acid a_i being aligned at position $A(i)$ regardless of the alignment of other amino acids. The first item generally refers to the probability of the template residue at position $A(i)$ mutating to the sequence residue a_i, and the probability of the sequence residue a_i occurring at the local structure environment of position $A(i)$ where the environment refers to both the local secondary structure type and solvent accessibility. The second item is the probability of two residues a_i and a_j simultaneously being aligned to two specific template positions $A(i)$ and $A(j)$. The remaining items refer to the probability of the multiple sequence residues simultaneously occurring at multiple specific template positions. Apart from first two items, the other items in the RHS of Equation 3.4 are often ignored because the parameter estimations of these items contain a high number of errors due to the lack of enough experimental data. Since $P(S|T,A)$ is the product of several items, we often use its negative logarithm form $f(S|T,A) = -\log P(S|T,A)$ for the sake of convenience and computation. The resulting f is the sum of several items and is called

the energy function in the threading context and the objective function in the optimization algorithm context.

Given a specific target sequence and a template, the general form of the energy function is

$$\begin{aligned} f(S|T,A) &= \sum_{j} f_1(j, A(j)) + \sum_{j_1,j_2} f_2(j_1, j_2, A(j_1), A(j_2)) + ... \\ &+ \sum_{j_1,j_2,...,j_M} f_M(j_1, j_2, ..., j_M, A(j_1), A(j_2), ..., A(j_M)) \end{aligned} \tag{3.5}$$

where $f_1(j, A(j))$ is the singleton score when the amino acid in sequence position j is placed to the template position $A(j)$; $f_2(j_1, j_2, A(j_1), A(j_2))$ represents the pairwise score when $A(j_1)$ and $A(j_2)$ are spatially nearby and the amino acids in the sequence position j_l are placed to the template positions $A(j_l)$ at the same time ($l = 1, 2$); and $f_3, f_4, ...$ denote the multi-body interaction scores that are often ignored in practice.

If only singleton scores and pairwise scores are considered, then the objective function in Equation 3.5 becomes

$$f(S|T,A) = \sum_{j} f_1(j, A(j)) + \sum_{j_1,j_2} f_2(j_1, j_2, A(j_1), A(j_2)) \tag{3.6}$$

For the sake of simplicity in this thesis, we employ Equation 3.6 as our objective function by default.

3.2 Computational Complexity

Various kinds of studies and implementations have been conducted for the protein threading problem [132, 68, 60, 102, 52, 109, 55]. However, most of the available prediction servers do not rigorously treat the second item of the right hand side of Equation 3.6 due to the computational complexity of searching for the optimal alignment between the target sequences to the templates. If Equation 3.6 is used as the objective function and gaps are allowed in the sequence to template alignment, then the threading problem is *NP*-hard [67, 1]. In their paper [1], Akutsu and Miyano conducted a comprehensive study concerning the

approximation of the protein threading problem. They have demonstrated that the protein threading problem is *MAX SNP*-hard, which means that no approximation algorithm can guarantee to generate a solution with any constant factor guarantee of accuracy within polynomial time unless *NP=P*. They also have shown that, in fact, the protein threading problem is much harder to approximate, by using an approximation preserving reduction from the DENSE-k-SUBGRAPH problem, for which only an $O(n^{-0.3885})$-approximation algorithm is known so far[1]. Besides the inapproximatibility results, Akutsu et al. also have proposed an $O(\sqrt{|E|})$-approximation algorithm for this problem where $|E|$ is the number of pairwise contacts in the template and given a constant factor approximation algorithm when the templates have a planar contact map graph [1]. However, not many templates have a planar contact graph, although Akutsu et al. argued that many templates could have a planar contact graph if only the pairwise contacts between the β-strands were considered.

Such complexity results suggest that the protein threading problem is very difficult. But the theoretical results are somewhat misleading and the approximation algorithms are not very useful in building real structure prediction programs. In this dissertation, we will demonstrate that, in practice, almost all protein threading instances can be solved within a reasonable (polynomial) time.

3.3 Threading Algorithms to Date

In this section, some representative approximation and exact algorithms for the protein threading problem will be introduced. One fact worth mentioning is that to our knowledge, no registered server in CAFASP3 [37] implements any of these algorithms.

3.3.1 Approximation Algorithms

1. **Monte Carlo Sampling and Simulated Annealing**: Bryant [14, 78] et al. used Simulated Annealing algorithm to search for the optimal alignment between the target sequence and the template. They began from one initial sequence-template

[1]Feige has improved it to $O(n^{-1/3+\epsilon})$ [31].

alignment and then used the Monte Carlo sampling technique to generate the next alignment from the current alignment. The energy function is calculated for each generated alignment, and the alignment is kept according to a certain probability calculated from its energy function value. Theoretically, the simulated annealing algorithm can converge to the optimal alignment (*i.e.*, minimal energy status) if enough computational time is given. But usually "enough computational time" is unaffordable if a long sequence is threaded.

2. **Interaction-Frozen Approximation** [9, 39, 40, 121]: This is a type of iterative algorithm. Given an initial alignment between the sequence and the template, one end of each contact (interaction) is fixed to the residue in the current alignment position. Then a dynamic programming algorithm is used to search for the next alignment based on the current "frozen" pairwise contact potential. This step is repeated for a number of times or until the next alignment is the same as the current alignment (*i.e.*, the algorithm converges.). The outstanding drawback of this method is that no convergence is guaranteed. That is, no optimal alignment is guaranteed.

3. **Recursive Dynamic Programming** [109]: This algorithm repeatedly uses a dynamic programming algorithm to match the target sequence to the template. At each iteration, the local alignment between the target sequences and the templates is searched by the dynamic programming algorithm. A segment of the target sequence is fixed onto a segment of the template if a significant similarity is attained. If the alignment position of one segment of sequence is fixed, then all pairwise contact potential involved with the residues in this segment is easy to handle. This is possible because one end of the pairwise contacts is frozen. The iteration is repeated until no significant similarity is found. Those unmatched segments of the target sequence are interpreted as gaps.

3.3.2 Exact Algorithms

To our knowledge, three different exact threading algorithms are proposed besides our linear programming approach to be described in the following chapters. Bryant and Lawrence

have designed an exhaustive search algorithm to enumerate all the possible sequence-template alignments, but their method is only suitable for a threading pair with a short target sequence and a template with a topologically simple contact graph.

Lathrop and Smith [68, 66] have proposed a branch-and-bound algorithm to solve the threading problem. The whole search space is split into many small subspaces by partitioning (branching) all the possible alignment positions of one template position into several intervals. For each subspace, Lathrop and Smith temporarily ignore all the pairwise contacts, search for the sequence-template alignment by a dynamic programming algorithm, and then estimate the bound of the objective function in the entire subspace. This is achieved by adding the pairwise score to the alignment score generated by the dynamic programming algorithm. This process is repeated, and some subspaces can be discarded during the search process based on the estimated bound. This algorithm runs fast for some practical instances, especially for self-threading. However, it has been reported that approximately 5% to 10% threading instances of their own test sets cannot be handled within a reasonable time [69, 68, 66].

Xu et al. have designed a divide-and-conquer algorithm which is used in their structure prediction computer program, PROSPECT-I [132, 129], based on an observation that if the cutoff distance of the pairwise contacts is fixed to $7\dot{A}$, then many (about three-quarters) of the template contact graphs are topologically simple. The idea is to split a template into two subsegments such that each subsegment is connected to as few external cores as possible, to recursively align each subsegment to the target sequence respectively, and finally, to merge the alignments of two subsegments to form an alignment for the whole segment. The key point is how to split the template to make the number of external cores, which the alignment of each generated subsegment to the target sequence depends on, as small as possible. The computational complexity is dominated by the alignment of one subsegment with the maximum number of external cores which is a small constant for topologically simple contact graphs. PROSPECT-I runs very fast for approximately three quarters of the templates. However, it runs very slowly (or runs out of memory on a 32-bit platform) on the remaining one-quarter templates with topologically complex contact graphs for long target sequences (≥ 400 residues).

Chapter 4

A New Approach to Protein Threading

In this chapter, our aim is to formulate the protein threading problem as an integer programming (IP) problem[1]. We first introduce the linear and integer programming approach, and describe the protein threading model and assumptions that are used in this thesis. It turns out that a straightforward IP formulation of the protein threading problem does not really work. We present three different integer program formulations in this chapter. We also prove the correctness of these formulations, and analyze and compare their strengths (tightnesses).

4.1 Introduction to Linear and Integer Programs

Linear and integer programming is a subfield of mathematical programming. A mathematical program tries to identify an extreme (*i.e.*, minimum or maximum) point of a function $f(x_1, x_2, ..., x_n)$ in a feasible region formed by a set of constraints, for example, $g(x_1, x_2, ..., x_n) \geq b$. When both the objective function f and the constraints are linear,

[1] After our first paper [128] appeared on the website of the 2003 Pacific Symposium in Biocomputing late in August 2002, another research group also proposed a very similar IP formulation to the protein threading problem in their technical report [133] in October 2002. In addition, our program RAPTOR attended CAFASP3 [37] starting from late May 2002 to September 2002.

we have a linear program (LP). When $x_1, \ldots, x_n$ are required to be integers, we have the integer linear programming (IP) problem. Linear and integer programming have been extensively used to solve many optimization problems derived from various application areas such as finance, economics, and planning [101, 115, 27]. A general form of a linear program with m constraints and n variables is

$$\min\{c^T x : Ax \geq b, x \geq 0, x \in R^n, A \in R^{m \times n}, b \in R^m, c \in R^n\}$$

where x is the vector of variables, c the cost vector, A the constraint matrix, and b the vector of allowed resources.

When the variable x is constrained to be 0 or 1, the linear program is often called a 0-1 integer program. 0-1 integer programs are used extensively to model many real-world decision-related problems, and many optimization problems which can be converted into decision-related problems. In this dissertation, we use the 0-1 integer programming approach to model the protein threading problem. A general form of a minimizing 0-1 integer program with m constraints and n variables is

$$\min\{c^T x : Ax \geq b, x \geq 0, x \in \{0,1\}^n, A \in R^{m \times n}, b \in R^m, c \in R^n\}$$

Linear programs can be solved within polynomial time whereas integer programs are NP-hard [98, 123]. For solving linear programs, there are three different kinds of methods: the Simplex method discovered by G. Dantzig in 1947 [24], the ellipsoid method by Russian mathematician Khachiyan in 1979 [61], and interior-point methods beginning with Karmarkar's in 1984 [58]. The Simplex method is a very fast algorithm for many practical problems, although it is not guaranteed to converge to the global optimal solution within polynomial time. For any variant of the Simplex method, there are always some examples contrived to make the algorithm take exponential time. The ellipsoid method is the first polynomial-time algorithm for linear programs, but it is very inefficient for most practical problems. The interior-point methods are the first practical polynomial time algorithm for linear programs. However, it is difficult to say in practice whether the Simplex method or the integer-point method is better [7]. The preferred method is problem-dependent.

The Simplex method solves a linear program by starting from a basic feasible solution, and then moving to another geometric neighbour repeatedly, until it arrives at the optimal

feasible solution. Geometrically, each vertex of the feasible solution region corresponds to a basic feasible solution. Each move is called an iteration. It can take thousands of iterations for the Simplex method to arrive at the optimal solution. But the cost of moving from one basic feasible solution to another one is very cheap. The path that the Simplex method goes through is always on the boundary of the feasible region. Conversely, the interior-point methods start from an interior point in the feasible region, and moves from an interior feasible solution to another one, and finally to the optimal one. Generally, it takes much fewer iterations for the interior-Point methods to converge to the optimum. But each iteration of the interior-point methods is rather expensive. In the last ten to twenty years, algorithms for solving linear programs have been improved greatly due to the development of sparse-matrix-related numerical computation, and the competition between the Simplex method and the interior-point methods [7].

In practical applications, the number of constraints (m) and the number of variables (n) are often very large, but the constraint matrix A is often quite sparse. That is, most entries are zero since each constraint often involves only a few variables. This allows the Simplex method and interior-point methods to converge very quickly by using special fast algorithms customized to work with sparse matrices. The current state-of-the-art linear program software packages allow us to deal with problems where the constraint matrix A has tens of million nonzero elements.

Although a polynomial-time algorithm for integer programs will not likely be found, many theories have been developed to understand them, including polyhedral and cutting plane theory. In the next chapter, an algorithm for solving integer programs is detailed.

4.2 Threading Assumptions

We represent the amino acid sequence, of length m, of a protein template by $t_1 t_2 \dots t_m$, and the query sequence, of length n, by $s_1 s_2 \dots s_n$.

Definition 4.2.1 *An alignment between the template and the sequence is a set of pairs* $(\hat{t_1}, \hat{s_1})$,...,$(\hat{t_i}, \hat{s_i})$,...,$(\hat{t_L}, \hat{s_L})$, *where* $L \leq m + n$, $\hat{t_1}\hat{t_2}...\hat{t_L}$ *is an expansion of* $t_1 t_2 \dots t_m$ *by inserting some gaps, and* $\hat{s_1}\hat{s_2}...\hat{s_n}$ *is an expansion of* $s_1 s_2 ... s_n$ *by inserting some gaps. For any pair* $(\hat{t_i}, \hat{s_i})$, *at most, one of* $\hat{t_i}$ *and* $\hat{s_i}$ *can be a gap.*

Definition 4.2.1 is a very general definition of an alignment, which can lead to a huge search space of feasible alignments. Some biological observations can be employed to give a more specific definition. In formulating the protein threading problem, we follow a few basic assumptions widely adopted by the protein threading community [132, 14, 78, 68]. We assume that:

1. Each template is parsed as a linear series of cores with connecting loops between adjacent cores. Each core is a conserved segment of an α-helix or β-sheet secondary structure among the template's homologs. Although the secondary structure is often conserved, insertions or deletions may occur at the two ends of a secondary structure. So, we only keep the most conserved part. Let $c_i = core(head_i, tail_i)$ denote all cores of one template, where $i = 1, 2, \ldots, M$ with M being the number of the cores, and $1 \leq head_1 \leq tail_1 < head_2 \leq tail_2 < \ldots < head_M \leq tail_M \leq m$. The regions between $tail_i$ and $head_{i+1}$, before $head_1$ and after $tail_M$ are loops. The length of core c_i is $len_i = tail_i - head_i + 1$. Let loc_i denote the sum of the length of all cores before c_i, that is, $loc_i = \sum_{j=1}^{i-1} len_j$.

2. When aligning a target protein sequence to a template, the alignment gaps are confined to only the loops of the template: the regions between cores or the two ends of the template. The biological justification is that the cores are so conserved that the chance of insertions or deletions within them is very small.

3. We consider only contacts (interactions) between the core residues. Generally it is believed that the interactions involving the loop residues can be ignored since their contribution to fold recognition is relatively insignificant. We say that an interaction exists between two residues if the spatial distance between their C_β atoms is below $7\dot{A}$, and they are at least 4 positions apart in the template sequence. An interaction exists between two cores if there exists at least one residue-residue interaction between the two cores.

As already discussed in Chapter 3, our threading scoring function consists of two types of items. One is the singleton score such as the environment fitness score E_s, mutation score E_m, and secondary structure compatibility score E_{ss}; the other is the gap penalty E_g

and pairwise interaction score E_p. By expanding the first item of the right hand side of Equation 3.6 to E_m, E_s and E_{ss}, and the second item to E_g and E_p, we have the following scoring function E:

$$E = W_m E_m + W_s E_s + W_p E_p + W_g E_g + W_{ss} E_{ss}, \tag{4.1}$$

where $W_m, W_s, W_p, W_g, W_{ss}$ are weight factors to be determined by training, which will be described in Chapter 7.

Global alignment and global-local alignment methods are employed to align the sequence to the template. The detailed description is given in Fischer et al.'s paper [36]. In the case that the template size is smaller than the sequence size, the whole template structure is aligned to the sequence. It is possible that the two ends of the sequence are not aligned by the template. The head gap penalty and tail gap penalty are used to penalize this kind of length mismatch. If the template size is larger than the query sequence size, it is possible that some cores at the two ends of the template cannot be aligned to the sequence. But we can always extend the sequence by adding some "artificial" amino acids to the two ends of the sequence to align all cores of the template to the (extended) sequence. All the scores involving those extended positions are set to be zero.

4.3 Threading Model

For each protein template 3D structure, we can build its detailed template contact map graph by modelling each template residue as a vertex and each residue-residue contact as an edge. If we formulate our integer program formulation based on the detailed template contact graph, there will be too many variables and non-zero elements in the constraint matrix, which will result in the LP solvers taking too much time to converge. Based on the assumption that no gaps are allowed within a core, we can simplify the template contact graph by merging all the vertices representing the residues in one core into a single vertex, that is, modelling each core as a vertex and adding one edge between two cores if there is at least one residue-residue contact between them. Formally, we define the following undirected contact graph:

Definition 4.3.1 *We use an undirected graph $CMG = (V, E)$ to denote the (simplified)*

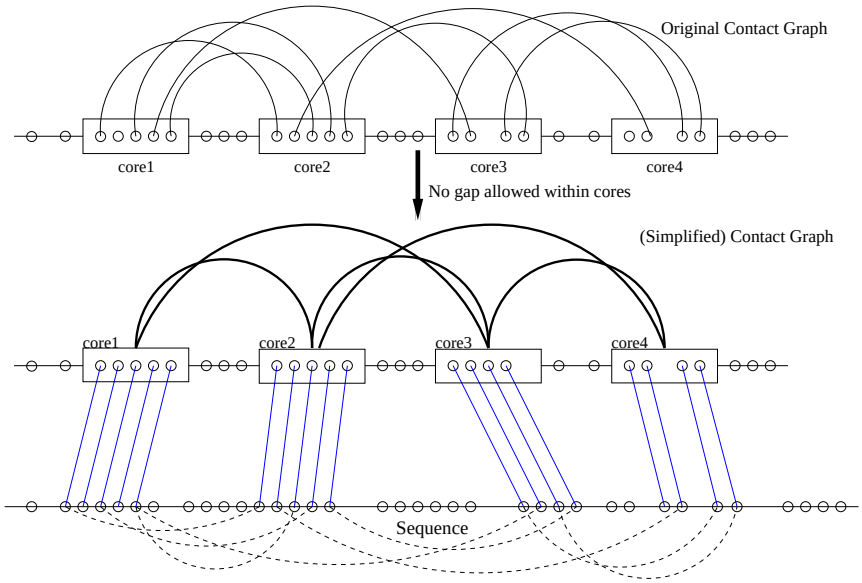

Figure 4.1: A template contact graph and an example of an alignment between one template and one sequence. A small circle represents one residue. One solid arc in the original contact graph indicates that its two end residues have an interaction. A dashed arc shows that if two sequence residues are aligned to two template residues having an interaction to each other, then the interaction score of these two sequence residues must be counted in the scoring function. The interaction score between two sequence segments is the sum of the interaction scores of two sequence residues which are aligned to two interacted template residues.

contact map graph of a protein template structure. Here, $V = \{c_1, c_2, ..., c_M\}$ *where* c_i *represents the* i^{th} *core, and* $E = \{(c_i, c_j) |$ *there is an interaction between* c_i *and* c_j*, or* $|i - j| = 1\}$.

Figure 4.1 presents an example of a contact graph.

For simplicity, when we say that core c_i is aligned to position s_j, we mean that core $c_i = (head_i, tail_i)$ is aligned to the sequence segment (s_j, s_{j+len_i-1}). In order to speed up the search, RAPTOR employs a knowledge-based filtering process proposed in Xu et al.'s paper [124] that indicates that certain alignments are *invalid*. Then we can construct a

bipartite graph to describe all the potential alignments between any core and any sequence position.

Definition 4.3.2 *Let B denote the alignment bipartite graph of one threading pair. Each core of the template corresponds to one vertex in B, labeled as $c_i (i = 1, 2, \ldots, M)$, and each residue in the query sequence corresponds to one vertex in B, labeled as $s_j (j = 1, 2, \ldots, n)$. The edges of B consist of all valid alignments between each core and each sequence position. The edges of B are also called the alignment edges.*

Obviously, in the alignment graph B, if two alignment edges of two cores are realized, then these two edges cannot cross. In the following definition, we give a formal definition of what it means for two alignment edges to cross or be in conflict.

Definition 4.3.3 *For any two different edges $e_1 = (c_{i_1}, s_{j_1})$ and $e_2 = (c_{i_2}, s_{j_2})$ in an alignment bipartite graph B, if $(loc_{i_1} - loc_{i_2})(s_{j_1} + loc_{i_2} - loc_{i_1} - s_{j_2}) \leq 0$, then we say e_1 and e_2 are in* conflict.

Let $D[i]$ denote all the valid query sequence positions that c_i can be aligned to. Let $R[i, j, l]$ denote all the valid alignment positions of c_j given that c_i is aligned to s_l. For any two edges (c_i, s_l), (c_j, s_k) in the alignment bipartite graph B, the following three properties are equivalent: (1) $k \in R[i, j, l]$; (2) (c_i, s_l) and (c_j, s_k) are not in conflict; and (3) $l \in R[j, i, k]$. Figure 4.2 illustrates an example of $D[i]$ and $R[i, j, l]$. We can now give a more practical definition of a valid alignment between the target sequences and the templates.

Definition 4.3.4 *An alignment between the template and the sequence is valid if:*

1. *It satisfies Definition 4.2.1;*
2. *Each core of the template is aligned to some position of the (extended) sequence. As mentioned before, global and global-local alignment are employed;*
3. *For any two different cores c_{i_1} and c_{i_2}, their two alignment edges do not conflict in the alignment graph, that is, if c_{i_j} is aligned to $s_{l_j} (j = 1, 2)$, then $s_{l_1} \in R[i_2, i_1, s_{l_2}]$ and $s_{l_2} \in R[i_1, i_2, s_{l_1}]$.*

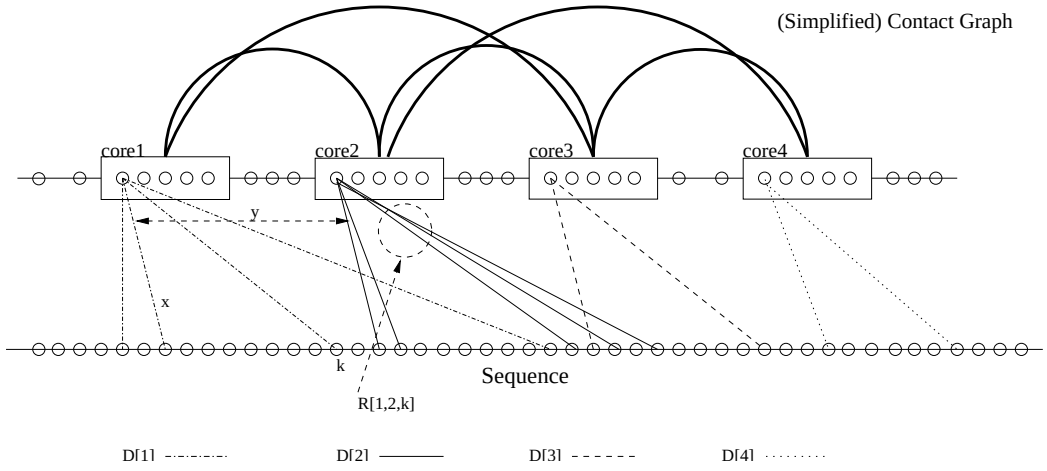

Figure 4.2: Example of $D[i]$ and $R[i, j, l]$. $R[1, 2, k]$ is the set of potential alignment positions of core 2 given core 1 is aligned to sequence position k. Core 1 has five residues which have to be aligned to the sequence based on our assumption in the "Alignment Model" subsection. Thus, the first two candidate alignment positions of core 2 are invalid if core 1 is aligned to position k in order to avoid overlap.

An optimal alignment is one that minimizes the objective function in Equation 4.1.

An alignment example is portrayed in Figure 4.1.

To facilitate the understanding of the formulations presented in the next section, we prove the following three lemmas. Lemma 4.3.1 describes the transitivity of the conflict relation, and Lemma 4.3.2 describes the transitivity of the non-conflict relation. These lemmas are also useful in the theoretical analysis of the formulations presented in the next chapter.

Lemma 4.3.1 *For any three different edges $e_r = (c_{i_r}, s_{j_r})$, $r = 1, 2, 3$, and $loc_{i_1} < loc_{i_2} < loc_{i_3}$, if e_1 conflicts with e_2 and e_2 conflicts with e_3, then e_1 conflicts with e_3.*

PROOF. For simplicity of notation, we will use l_i for loc_i.

$$
\begin{aligned}
& (l_{i_1} - l_{i_3})(s_{j_1} + l_{i_3} - l_{i_1} - s_{j_3}) \\
= \; & (l_{i_1} - l_{i_2} + l_{i_2} - l_{i_3})(s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2} + s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3}) \\
= \; & (l_{i_1} - l_{i_2})(s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2}) + (l_{i_2} - l_{i_3})(s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3}) + \\
& (l_{i_1} - l_{i_2})(s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3}) + (l_{i_2} - l_{i_3})(s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2}) \\
\leq \; & 0 + 0 + (l_{i_1} - l_{i_2})(s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3}) + (l_{i_2} - l_{i_3})(s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2}) \\
= \; & (l_{i_1} - l_{i_2})(l_{i_2} - l_{i_3})((s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3})/(l_{i_2} - l_{i_3}) \\
& + (s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2})/(l_{i_1} - l_{i_2})) \\
\leq \; & 0
\end{aligned}
$$

□

Lemma 4.3.2 *For any three different edges $e_r = (c_{i_r}, s_{j_r})$, $r = 1, 2, 3$ and $loc_{i_1} < loc_{i_2} < loc_{i_3}$, if e_1 does not conflict with e_2 and e_2 does not conflict with e_3, then e_1 does not conflict with e_3.*

PROOF. For simplicity of notation, we will use l_i for loc_i.

$$(l_{i_1} - l_{i_3})(s_{j_1} + l_{i_3} - l_{i_1} - s_{j_3})$$

$$
\begin{aligned}
&= (l_{i_1} - l_{i_2} + l_{i_2} - l_{i_3})(s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2} + s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3}) \\
&= (l_{i_1} - l_{i_2})(s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2}) + (l_{i_2} - l_{i_3})(s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3}) + \\
&\quad (l_{i_1} - l_{i_2})(s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3}) + (l_{i_2} - l_{i_3})(s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2}) \\
&\geq 0 + 0 + (l_{i_1} - l_{i_2})(s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3}) + (l_{i_2} - l_{i_3})(s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2}) \\
&= (l_{i_1} - l_{i_2})(l_{i_2} - l_{i_3})((s_{j_2} + l_{i_3} - l_{i_2} - s_{j_3})/(l_{i_2} - l_{i_3}) + \\
&\quad (s_{j_1} + l_{i_2} - l_{i_1} - s_{j_2})/(l_{i_1} - l_{i_2})) \\
&> 0
\end{aligned}
$$

□

Lemma 4.3.3 *For any three different edges $e_r = (c_{i_r}, s_{j_r})$, $r = 1, 2, 3$ and $loc_{i_1} < loc_{i_2} < loc_{i_3}$, if e_1 conflicts with e_3, then e_2 conflicts with e_3 or e_2 conflicts with e_1.*

Proof. This lemma follows from Lemma 4.3.2. □

4.4 Integer Program Formulation

In our formulation, we employ two kinds of indicator variables. Let $x_{i,l}$ be a 0-1 variable such that $x_{i,l} = 1$ if and only if core c_i is aligned to position s_l, that is, edge (c_i, s_l) in B is realized. Similarly, for any $(c_{i_1}, c_{i_2}) \in E(CMG)$, let $y_{(i_1,l_1),(i_2,l_2)}$ indicate the pairwise interactions between x_{i_1,l_1} and x_{i_2,l_2} if the two edges $(c_{i_1}, s_{l_1}), (c_{i_2}, s_{l_2})$ do not conflict. $y_{(i_1,l_1),(i_2,l_2)} = 1$ if and only if $x_{i_1,l_1} = 1$ and $x_{i_2,l_2} = 1$. We say $y_{(i_1,l_1),(i_2,l_2)}$ is generated by x_{i_1,l_1} and x_{i_2,l_2}. We call x the alignment variable and y the interaction variable.

Now, we formulate the objective function of the protein threading problem as follows:

$$
\begin{aligned}
f &= W_m E_m + W_s E_s + W_p E_p + W_g E_g + W_{ss} E_{ss}, where \\
E_m &= \sum_{i=1}^{M} \sum_{l \in D[i]} [x_{i,l} \sum_{r=0}^{len_i - 1} Mutation(head_i + r, l + r)]; \\
E_s &= \sum_{i=1}^{M} \sum_{l \in D[i]} [x_{i,l} \sum_{r=0}^{len_i - 1} Fitness(head_i + r, j + r)];
\end{aligned}
$$

$$
\begin{aligned}
E_{ss} &= \sum_{i=1}^{M} \sum_{l \in D[i]} [x_{i,l} \sum_{r=0}^{len_i - 1} SS(head_i + r, j + r)]; \\
E_p &= \sum_{(c_i, c_j) \in E(CMG)} \sum_{l \in D[i]} \sum_{k \in R[i,j,l]} y_{(i,l),(j,k)} P(i,j,l,k); \\
P(i,j,l,k) &= \sum_{u=0}^{len_i - 1} \sum_{v=0}^{len_j - 1} \delta(t_{head_i + u}, t_{head_j + v}) Pair(l + u, k + v); \\
E_g &= \sum_{i=1}^{M} \sum_{l \in D[i]} \sum_{k \in R[i,i+1,l]} y_{(i,l),(i+1,k)} G(i,l,k).
\end{aligned}
$$

In this objective function, the singleton score items E_m, E_s, E_{ss} are the expansion of the first item of the right hand side of Equation 3.6; the pairwise score items E_p and E_g are the expansion of the second item of this equation. *Mutation()* is the log-odds of two residues mutating; *Fitness()* is the log-odds of one residue being in one environment; *Pair()* is the log-odds of two residues being spatially nearby; *SS()* measures the compatibility of the predicted sequence secondary structure and the template secondary structure. $\delta(t_u, t_v) = 1$ if there is an interaction between the residues at position u and v in the template; otherwise it is 0. $G(i,l,k)$ is the sequence alignment score of aligning the segment between core i and core $i+1$ to the sequence segment from position l to k. It is $G(i,l,k)$ that contains the gap penalty to penalize the alignment gap. Because we do not consider the residue-residue contacts involving the loop residues, $G(i,l,k)$ can be computed by a dynamic programming algorithm. In order to improve the computational efficiency, we can reuse the dynamic programming table for the calculation of $G(i,l,k)$ to compute $G(i,l,k_1)$ for any $k_1 < k$. The parameters used to calculate *Mutation()*, *Fitness()*, *SS()*, *Pair()* and *G()* are discussed in Section 7.1.

The constraint set is as follows:

$$
\begin{aligned}
\sum_{j \in D[i]} x_{i,j} &= 1, i = 1, 2, \ldots, M; && (4.2) \\
\sum_{l \geq l_0, l \in D[i]} x_{i,l} + \sum_{k \in D[i+1] - R[i,i+1,l_0]} x_{i+1,k} &\leq 1, \ \textit{for all } l_0 \in D[i]; && (4.3) \\
\sum_{k \in R[i,j,l]} y_{(i,l),(j,k)} &\leq x_{i,l}, \ \textit{for all } l \in D[i]; && (4.4)
\end{aligned}
$$

$$\sum_{l \in R[j,i,k]} y_{(i,l),(j,k)} \leq x_{j,k}, \text{ for all } k \in D[j]; \quad (4.5)$$

$$1 + \sum_{k \in R[i,j,l]} y_{(i,l),(j,k)} \geq x_{i,l} + \sum_{k \in R[i,j,l]} x_{j,k}, \text{ for all } l \in D[i]; \quad (4.6)$$

$$1 + \sum_{l \in R[j,i,k]} y_{(i,l),(j,k)} \geq x_{j,k} + \sum_{l \in R[j,i,k]} x_{i,l}, \text{ for all } k \in D[j]; \quad (4.7)$$

$$x_{i,j} \in \{0,1\}; \quad (4.8)$$

$$y_{(i,l)(j,k)} \in \{0,1\}. \quad (4.9)$$

Constraint 4.2 indicates that one core can be aligned to a unique sequence position. Constraint 4.3 forbids conflicts between the alignments of two adjacent cores. Therefore, based on the transitivity of nonconflict (see Lemma 4.3.2), this constraint guarantees that there are no conflicts between the alignments of any two cores. Constraints 4.4 and 4.5 tell us that at most one of all the interaction variables generated by one x variable can be realized only if the x variable is realized. Constraints 4.6 and 4.7 enforce that if two x variables are realized simultaneously, then their generating y variable should be realized. Constraints 4.8 and 4.9 restrict x and y to be either 0 or 1.

There is another set of more obvious constraints which can replace Constraint 4.3–4.7. They are:

$$x_{i,l} + x_{i+1,k} \leq 1, \text{ for all } k \in D[i+1] - R[i,i+1,l]; \quad (4.10)$$

$$y_{(i,l)(j,k)} \leq x_{i,l}, \text{ for all } k \in R[i,j,l], (c_i,c_j) \in E(CMG); \quad (4.11)$$

$$y_{(i,l)(j,k)} \leq x_{j,k}, \text{ for all } l \in R[j,i,k], (c_i,c_j) \in E(CMG); \quad (4.12)$$

$$y_{(i,l)(j,k)} \geq x_{i,l} + x_{j,k} - 1, (c_i,c_j) \in E(CMG). \quad (4.13)$$

Constraint 4.10 forbids the conflict between the alignments of two adjacent cores. Similarly, it guarantees that there is no conflict between the alignments of any two cores. Constraints 4.11 to 4.13 guarantee that one interaction variable is realized if and only if its two generating x variables are realized at the same time. Constraints 4.10 to 4.13 can be inferred from Constraints 4.3 to 4.7. The converse inference does not hold. Therefore, Constraints 4.10-4.13 are weaker than Constraints 4.3-4.7.

After carefully examining the problem, we find yet another set of constraints (Constraint 4.14 and 4.15) from which the previous two constraint sets can be inferred (See Lemma

4.4.1 and 4.4.2).

$$\sum_{k \in R[i,j,l]} y_{(i,l)(j,k)} = x_{i,l}, (c_i, c_j) \in E(CMG); \quad (4.14)$$

$$\sum_{l \in R[j,i,k]} y_{(i,l)(j,k)} = x_{j,k}, (c_i, c_j) \in E(CMG). \quad (4.15)$$

Constraint 4.14 and 4.15 imply that one x variable is realized if and only if one of the y variables generated by it is realized. These two are the strongest constraints. Experimental results show that our program with Constraint 4.14 and 4.15 (combined with Constraint 4.2, 4.8 and 4.9) runs significantly faster. Our program RAPTOR uses Constraint 4.14 and 4.15 by default.

To solve the previous integer programs, we first relax the integral constraints to linear constraints. That is, we relax Constraint 4.8 and 4.9 to the following two constraints:

$$x_{i,j} \geq 0, j \in D[i], i = 1, 2, \ldots, M; \quad (4.16)$$

$$y_{(i,l)(j,k)} \geq 0, \forall l \in D[i], k \in D[j], i, j = 1, 2, \ldots, M. \quad (4.17)$$

Let *CS1* denote the constraint set formed by Constraint 4.2-4.7, 4.16 and 4.17; *CS2* the constraint set formed by Constraint 4.2, 4.10-4.13, 4.16, and 4.17; *CS3* the constraint set formed by Constraint 4.2, 4.14-4.15 and 4.16, 4.17. Then we have the following lemmas:

Lemma 4.4.1 *Constraint set CS1 is implied by CS3.*

PROOF. Note that for all i, $R[i,j,l] \subseteq D[j]$. We can easily see that Constraint 4.4-4.7 are implied by Constraint 4.14 and 4.15. The following calculation shows that Constraint 4.3 is also implied by *CS3*.

$$\begin{aligned}
& \sum_{l \geq l_0, l \in D[i]} x_{i,l} + \sum_{k \in D[i+1] - R[i,i+1,l_0]} x_{i+1,k} \\
= & \sum_{l \geq l_0, l \in D[i]} x_{i,l} + \sum_{k \in D[i+1] - R[i,i+1,l_0]} \Big(\sum_{l \in R[i+1,i,k]} y_{(i,l)(i+1,k)} \Big) \\
\leq & \sum_{l \geq l_0, l \in D[i]} x_{i,l} + \sum_{l \in D[i], l < l_0} \Big(\sum_{k \in R[i,i+1,l]} y_{(i,l)(i+1,k)} \Big) \\
\leq & \sum_{l \geq l_0, l \in D[i]} x_{i,l} + \sum_{l \in D[i], l < l_0} x_{i,l}
\end{aligned}$$

$$= \sum_{l \in D[i]} x_{i,l}$$
$$= 1$$

□

Lemma 4.4.2 *Constraint set CS2 is implied by CS1.*

PROOF. Constraint 4.10 is implied by Constraint 4.3 and 4.16. Constraint 4.11-4.12 are implied by Constraint 4.4-4.5 and 4.17. Constraint 4.13 is implied by Constraint 4.6-4.7 and 4.11-4.12. □

We also have the following lemma about the correctness of our IP formulation.

Lemma 4.4.3 *Each integral solution of CS1, CS2, CS3 corresponds to a valid alignment between the template and the sequence.*

PROOF. First, we prove that any integral solution of *CS2* corresponds to a valid alignment. According to Constraint 4.10, for any two adjacent cores i, $i+1$, if they are aligned to sequence positions l, k respectively, then these two alignment edges (c_i, s_l) and (c_{i+1}, s_k) do not conflict. Based on the transitivity of nonconflict (see Lemma 4.3.2), the two alignment edges of any two cores cannot conflict; that is, any integral solution of *CS2* corresponds to a valid alignment.

Based on Lemma 4.4.2 and 4.4.1, Constraint 4.10 is implied by *CS1* and *CS3*; therefore, each integral solution of *CS2* or *CS3* corresponds to a valid alignment. □

Lemma 4.4.4 *There exists an instance of protein threading problem such that there is one solution of CS1 and CS2 which violates the constraints of CS3.*

PROOF. Consider the template with only two cores c_1 and c_2. Assume $D[1] = \{j_1, j_2\}$, $D[2] = \{j_3, j_4\}$, $R[1, 2, j_1] = D[2]$ and $R[1, 2, j_2] = \{j_4\}$. Then this instance has four x variables: x_{1,j_1}, x_{1,j_2}, x_{2,j_3}, x_{2,j_4} and three y variables: $y_{(1,j_1),(2,j_3)}$, $y_{(1,j_1),(2,j_4)}$, $y_{(1,j_2),(2,j_4)}$. Let all x variables be 0.5 and all y variables be 0.25. It is easy to verify that *CS1* and *CS2* is satisfied but *CS3* is violated. □

So far we have presented three kinds of linear (integer) program formulations (the objective function combined with each of *CS1*, *CS2* and *CS3*) to formulate the sequence-template alignment problem. We have also proved that the constraint set *CS3* is the strongest when the integrality constraints on x and y variables are relaxed to allow real values between 0 and 1. Experimental results demonstrate that our program using *CS3* runs much faster than using *CS1* or *CS2*. RAPTOR uses *CS3* by default.

Chapter 5

Theoretical Analysis of The IP Formulation

This chapter will further analyze the integer program formulation proposed in Chapter 4 from the perspective of polyhedral combinatorics. We attempt to find two types of well-known cuts to strengthen our formulation. The result of the cutting plane analysis is that these cuts are already implied in our constraint set. Therefore, it is unnecessary to add them as extra constraints. In this chapter, we first introduce two canonical methods to solve integer programs: the branch-and-bound method and the branch-and-cut method. Then we prove that the cutting planes sought after are already redundant with respect to our original constraint set. Finally, we describe our experiments on the integrality of the optimal linear solutions of our LP formulations derived from real instances.

5.1 Solving Integer Programs

In Chapter 4, we have introduced linear and integer programs as well as several methods for solving linear programs. In this section, we will discuss how to solve integer programs. Historically, the first method to solve integer programs was the branch-and-cut method or cutting-plane method [98]. In the last twenty to thirty years, the most effective method for integer programs has proved to be the branch-and-bound method [7]. This method splits the original problem into two (or several) subproblems and solves each subproblem

recursively. However, in the past ten years, the cutting-plane method has made a resurgence in the form of facets and polyhedral characterizations [7].

5.1.1 Branch-and-Bound Method

The branch-and-bound method [123] first solves the linear version of an integer program, and then tests if the linear solution $\hat{x}$ is integral or not. If the solution is integral, then the optimal integral solution is found. Otherwise, it chooses a fractional component of vector $\hat{x}$, say $\hat{x}_i$, by a certain strategy, and splits the original problem into two subproblems at x_i. There are various kinds of selection strategies to determine at which fractional component to branch. Two new constraints $x_i \geq \lceil \hat{x}_i \rceil$ and $x_i \leq \lfloor \hat{x}_i \rfloor$ are added into the two subproblems respectively. The linear versions of these two subproblems are solved again. If one generates an integral solution, then the algorithm updates the best-so-far integral solution and its corresponding best-so-far objective value. Otherwise, it compares the objective value to the best-so-far objective value. If the objective value is no better than the best-so-far objective value, then the algorithm discards (prunes) this subproblem. If the subproblem is not discarded, then it is split and solved recursively.

5.1.2 Branch-and-Cut Method

The branch-and-cut method [98] can be regarded as a generalization of the branch-and-bound method. It will not only employ all the procedures of the branch-and-bound method, but also add some cutting planes into the original problem or some subproblems before conducting the branch operation. A cutting plane is a hyperplane which can separate the feasible region from a given point outside the region. The only difference between the branch-and-cut method and the branch-and-bound method is that the branch-and-cut method looks for some cuts first if the linear solution is not integral. The cuts are used to separate the fractional solution from the feasible integral solutions. If the specific cuts are found, then the algorithm adds these cuts into the relaxed linear program and solves the resultant linear program again. Otherwise, the algorithm takes the same procedure as the branch-and-bound method. There are several strategies differing in the time when the cuts are added. Cuts can be added at the very beginning, that is, before solving the linear

relaxation of the original problem; cuts can also be added to some chosen subproblems. Generally, in order to improve efficiency, cuts are not added to each subproblem. The key factors for the success of the branch-and-cut method are that the chosen cuts can be found inexpensively and that they are helpful to strengthen the original problem, that is, the cuts are strong constraints. Therefore, the effectiveness and efficiency of the branch-and-cut method is also problem-dependent, just like the performance of the Simplex method and the Interior-Point method.

5.2 Theoretical Analysis

When we use the branch-and-cut method to solve our integer program, we also attempt to recognize some effective cutting planes (also called valid inequalities or cuts) which can help to strengthen the integer program. The threading alignment problem can be treated as the maximum independent set (MIS) problem (see Section 5.2.1). For MIS problems, there are two kinds of well-known valid inequalities that might be used to add strength to the formulation. However, we will see that they are already implied by the constraints of our formulation due to the special structure of our problem. This suggests that our formulation is very strong and effective. We will also prove that for a special type of contact graph, the optimal linear solutions of the relaxed linear programs are integral.

5.2.1 Seeking Cutting Planes

Before presenting the detailed justification of our results, we first prepare some definitions and formally prove some simple lemmas. For the sake of simplicity, we continue to use the notation used in Chapter 4. In the integer program formulation, each x variable has a one-to-one mapping to the corresponding alignment edge in B. We have defined "conflict" for the alignment edges in B and proved several lemmas concerning it. In this chapter, for the sake of convenience, we redefine "conflict" for x variables and list several related lemmas although the corresponding versions are in Chapter 4, where the detailed proofs are given. In the following definitions and lemmas, Definition 5.2.1 corresponds to Definition 4.3.3; Lemma 5.2.2 corresponds to Lemma 4.3.1; Lemma 5.2.3 corresponds to Lemma 4.3.2; and Lemma 5.2.5 corresponds to Lemma 4.3.3.

Definition 5.2.1 *For any two different x variables x_{i_1,j_1} and x_{i_2,j_2}, if $(loc_{i_1} - loc_{i_2})(j_1 + loc_{i_2} - loc_{i_1} - j_2) \leq 0$, then we say that they are in* conflict. *Obviously, $j_2 \notin R[i_1, i_2, j_1]$ and $j_1 \notin R[i_2, i_1, j_2]$, if x_{i_1,j_1} and x_{i_2,j_2} are in conflict.*

Lemma 5.2.1 *For any two variables x_{i,l_0} and x_{j,k_0} $(i \neq j)$, if x_{i,l_0} and x_{j,k_0} conflict, then for all $l \geq l_0, k \leq k_0$, $x_{i,l}$ and $x_{j,k}$ conflict.*

PROOF. Assume $i < j \Rightarrow loc_i < loc_j$, then $l_0 - k_0 + loc_j - loc_i \geq 0$. $\forall l \geq l_0, k \leq k_0$, we have $l - k + loc_j - loc_i \geq l_0 - k_0 + loc_j - loc_i \geq 0$. Therefore, $(loc_i - loc_j)(l - k + loc_j - loc_i) \leq 0$. □

Lemma 5.2.2 *For any three different x variables x_{i_r,j_r} where $r = 1, 2, 3$ and $i_1 < i_2 < i_3$, if x_{i_1,j_1} conflicts with x_{i_2,j_2} and x_{i_2,j_2} conflicts with x_{i_3,j_3}, then x_{i_1,j_1} conflicts with x_{i_3,j_3}.*

Lemma 5.2.3 *For any three different x variables x_{i_r,j_r} where $r = 1, 2, 3$ and $i_1 < i_2 < i_3$, if x_{i_1,j_1} does not conflict with x_{i_2,j_2} and x_{i_2,j_2} does not conflict with x_{i_3,j_3}, then x_{i_1,j_1} does not conflict with x_{i_3,j_3}.*

Lemma 5.2.4 *For any three different x variables x_{i_r,j_r} where $r = 1, 2, 3$, $i_1 < i_2 = i_3$ and $j_2 < j_3$, if x_{i_1,j_1} does not conflict with x_{i_2,j_2}, then x_{i_1,j_1} does not conflict with x_{i_3,j_3}.*

Lemma 5.2.5 *For any three different edges x_{i_r,j_r} where $r = 1, 2, 3$, and $i_1 < i_2 < i_3$, if x_{i_1,j_1} conflicts with x_{i_3,j_3}, then x_{i_2,j_2} conflicts with either x_{i_1,j_1} or x_{i_3,j_3}.*

Lemma 5.2.6 *For any three different edges x_{i_r,j_r} where $r = 1, 2, 3$, $j_2 < j_3$, and $i_1 < i_2 = i_3$, if x_{i_1,j_1} conflicts with x_{i_3,j_3}, then x_{i_2,j_2} conflicts with x_{i_1,j_1}.*

PROOF. This lemma follows from Lemma 5.2.4. □

We also need some definitions for the conflict involving y variables.

Definition 5.2.2 *For any $x_{i,l}$ and $y_{(i_1,l_1)(i_2,l_2)}$, $(i,l) \neq (i_1,l_1)$, $(i,l) \neq (i_2,l_2)$, we say $x_{i,l}$ conflicts with $y_{(i_1,l_1)(i_2,l_2)}$, if $x_{i,l}$ conflicts with either x_{i_1,l_1} or x_{i_2,l_2}.*

Definition 5.2.3 *For any two different $y_{(i_1,l_1)(i_2,l_2)}$ and $y_{(i_3,l_3)(i_4,l_4)}$, we say they conflict if either x_{i_1,l_1} or x_{i_2,l_2} conflicts with $y_{(i_3,l_3)(i_4,l_4)}$.*

Let X denote the set of all x variables and Y denote the set of all y variables. Let $X(c_i)$ denote the set of x variables derived from core c_i. The following two definitions model the conflict relationship of the x variables and of the y variables as graphs.

Definition 5.2.4 *The x-variable graph $G_x = (V(G_x), E(G_x))$ is generated from X by letting $V(G_x) = X$, and $E(G_x) = \{(x^1, x^2) | x^1, x^2 \in X$ and $x^1 conflicts with x^2$.*

Definition 5.2.5 *The y-variable graph $G_y = (V(G_y), E(G_y))$ is generated from Y by letting $V(G_y) = Y$ and $E(G_y) = \{(y^1, y^2) | y^1, y^2 \in Y$ and $y^1 conflicts with y^2\}$.*

For any feasible integral solution of our integer program, the set of x variables with value 1 form a maximal independent set of G_x. Similarly, the set of y variables with value 1 form a maximal independent set of G_y. So our threading alignment problem is transformed into the problem of finding a maximal independent set which minimizes the scoring function. For the maximal independent set problem, we have two kinds of well-known valid inequalities to strengthen the integer program [19]. One is the clique inequality, that is, at most one edge of a clique can be in the maximal independent set. The other is the odd hole (A hole is a chordless cycle of length at least four) inequality, that is, at most $\lfloor \frac{|H|}{2} \rfloor$ edges of an odd cycle H ($|H| \geq 5$) can be in the maximal independent set. Generally, the number of these two kinds of inequalities is exponential so they cannot be incorporated into the integer program formulation explicitly. The branch-and-cut method makes use of them by dynamically incorporating some cuts that are violated by the current optimal linear solution if these violated inequalities can be recognized within a reasonable time. In the following paragraphs, we will prove that all the x-clique inequalities are already implied by the constraints of the integer program (Theorem 5.2.9) Also, there is no odd hole in G_x (Theorem 5.2.10). Several types of simple y-clique inequalities are also implied. Therefore, it is unnecessary for us to add the x-clique inequalities during the solution process of the branch-and-cut method. Before proving Theorem 5.2.9 and Theorem 5.2.10, we need to prove some lemmas.

Lemma 5.2.7 *Given three cores c_{i_1}, c_{i_2}, and c_{i_3} with $i_1 < i_2 < i_3$ and two variable sets $X^1 = \{x_{i_1,l}, l \geq l_0\}$ and $X^3 = \{x_{i_3,k}, k \leq k_0\}$, suppose that any two variables $x_{i_1,l} \in X^1$ and $x_{i_3,k} \in X^3$ conflict. Then each variable in $X(c_{i_2})$ conflicts with either all variables in X^1 or all in X^3.*

PROOF. For each $x_{i_2,r}$ in $X(c_{i_2})$, because x_{i_1,l_0} conflicts with x_{i_3,k_0}, based on Lemma 5.2.5, we see that $x_{i_2,r}$ conflicts with x_{i_1,l_0} or x_{i_3,k_0}. Without loss of generality, assume $x_{i_2,r}$ conflicts with x_{i_1,l_0}. Then $(r - l_0 + loc_{i_1} - loc_{i_2}) \leq 0$ (because $loc_{i_2} > loc_{i_1}$). For each $x_{i_1,l}$, $l \geq l_0$, we have $(r - l + loc_{i_1} - loc_{i_2}) \leq (r - l_0 + loc_{i_1} - loc_{i_2}) \leq 0$. Therefore, $x(i_2, r)$ conflicts with $x_{i_1,l}$ for all $l \geq l_0$. □

Lemma 5.2.7 states that for any two nonadjacent cores c_i, c_j and two different x variable sets $X^1 \subseteq X(c_i)$, $X^2 \subseteq X(c_j)$, if any two variables in $X^1 \cap X^2$ conflict with each other, then any x variable from any core in segment $c_{i+1},...,c_{j-1}$ conflicts with either all variables in X^1 or all variables in X^2. Lemma 5.2.7 can be regarded as a generalization of Lemma 5.2.5.

In order to prove Theorem 5.2.9, that is, any x clique inequality is already implied by the constraint set of our formulation, we first prove a simple case: any x clique inequality involving with x variables of only two cores is implied by the constraint set. We then generalize the simple case. The simple case is shown in Lemma 5.2.8.

Lemma 5.2.8 *Each x-clique inequality involving x variables of only two cores is implied by Constraint 4.2, 4.14-4.15 and 4.16, 4.17.*

PROOF. Suppose that the maximal clique of G_x consists of all x-variables from two sets $X(c_i)$ and $X(c_j)$ $(i < j)$. We use induction on the value of $j - i$. Let $X_i = \{x_{i,l}, l \geq l_0\} \subseteq X(c_i)$ and $X_j = \{x_{j,k}, k \leq k_0\} \subseteq X(c_j)$ be the variables contained in the clique. We need to show $\sum_{x \in X_i} x + \sum_{x \in X_j} x \leq 1$. Obviously we have k_0 is less than the minimum element in $R[i, j, l_0]$.

First we prove the case $j - i = 1$. From Constraints 4.14 and 4.15, we have

$$\sum_{l \geq l_0} x_{i,l} + \sum_{k \leq k_0} x_{i+1,k} = \sum_{l \geq l_0} \sum_{k_1 \in R[i,i+1,l]} y_{(i,l)(i+1,k_1)} + \sum_{k \leq k_0} x_{i+1,k}$$

$$
\begin{aligned}
&\leq \sum_{k_1>k_0} \sum_{l\geq l_0, l\in R[i+1,i,k_1]} y_{(i,l)(i+1,k_1)} + \sum_{k\leq k_0} x_{i+1,k} \\
&\leq \sum_{k_1>k_0} \sum_{l\in R[i+1,i,k_1]} y_{(i,l)(i+1,k_1)} + \sum_{k\leq k_0} x_{i+1,k} \\
&= \sum_{k_1>k_0} x_{i+1,k_1} + \sum_{k\leq k_0} x_{i+1,k} \\
&= \sum_{k\in D[i+1]} x_{i+1,k} \\
&= 1
\end{aligned}
$$

Now assume that when $j - i \leq m$, the proposition holds. We will prove it for the case $j - i = m + 1$. For any r between i and j, obviously, $r - i \leq m$ and $j - r \leq m$. Based on Lemma 5.2.7, for all $x_{r,l}$, $x_{r,l}$ must conflict with either all $x_{i,l}$, $(l \geq l_0)$ or all $x_{j,k}$, $(k \leq k_0)$. Let $X_r^1 = \{x_{r,l_r} | x_{r,l_r}$ conflicts with all edges in $X_i\}$ and $X_r^2 = \{x_{r,l_r} | x_{r,l_r}$ conflicts with all edges in $X_j\}$. Then $X(c_r) \subseteq (X_r^1 \cup X_r^2)$ and $\sum_{x\in X_r^1} x + \sum_{x\in X_r^2} x \geq \sum_{x\in X(c_r)} x \geq 1$. X_i, X_r^1 form a clique and X_j, X_r^2 form a clique. This yields,

$$
\begin{aligned}
\sum_{l\geq l_0} x_{i,l} + \sum_{k\leq k_0} x_{j,k} &= (\sum_{l\geq l_0} x_{i,l} + \sum_{x\in X_r^1} x) + (\sum_{k\leq k_0} x_{j,k} + \sum_{x\in X_r^2} x) - (\sum_{x\in X_r^1} x + \sum_{x\in X_r^2} x) \\
&\leq 1 + 1 - 1 \\
&= 1
\end{aligned}
$$

This completes the proof of the lemma by induction. □

Theorem 5.2.9 *Each x-clique inequality is implied by Constraint 4.2, 4.14-4.15 and 4.16, 4.17.*

PROOF. For a given maximal clique of G_x, we use induction on the number of cores.

In the base case, the maximal clique consists of variables from just two cores. Then the proposition holds according to Lemma 5.2.8.

Now assume the proposition holds if the maximal clique is formed by x variables from no more than i cores. Consider a maximal clique formed by the x variables of core c_{q_1}, c_{q_2}, ..., c_{q_i}, $c_{q_{i+1}}$. Let $X_{clique}(c_r)$ denote the x variables of cores c_r for all r in $\{q_1, q_2, ..., q_{i+1}\}$ contained in this maximal clique. Based on Lemma 5.2.7 and Lemma 5.2.2, for any $x \in X(c_{q_i})$, at least one of the following two statements holds:

1. x conflicts with all variables in sets $X_{clique}(c_r)$ for all r in $\{q_1, q_2, ..., q_{i-1}\}$. Let $X^1(c_{q_i})$ denote the set of this kind of x, then $X_{clique}(c_{q_1})$, $X_{clique}(c_{q_2})$, ..., $X_{clique}(c_{q_{i-1}})$ and $X^1(c_{q_i})$ form a clique;

2. x conflicts with all variables in set $X_{clique}(c_{q_{i+1}})$. Let $X^2(c_{q_i})$ denote the set of this kind of x, then $X^2(c_{q_i})$ and $X_{clique}(c_{q_{i+1}})$ form a clique.

Obviously $X^1(c_{q_i}) \cup X^2(c_{q_i}) = X(c_{q_i})$ and $X^1(c_{q_i}) \cap X^2(c_{q_i}) = X_{clique}(c_{q_i})$ (otherwise, this clique will not be maximal). Let $S(Z)$ denote the sum of x variables in the set Z. We have

$$\begin{aligned}
\sum_{r=1}^{i+1} S(X_{clique}(c_{q_r})) &= \sum_{r=1}^{i-1} S(X_{clique}(c_{q_r})) + S(X_{clique}(c_{q_i})) + S(X_{clique}(q_{i+1})) \\
&= \sum_{r=1}^{i-1} S(X_{clique}(c_{q_r})) + S(X^1(c_{q_i})) + S(X^2(c_{q_i})) \\
&+ S(X_{clique}(c_{q_{i+1}})) - S(X(c_{q_i})) \\
&\leq 1 + 1 - 1 \\
&= 1
\end{aligned}$$

This concludes the proof by induction of the lemma. □

So far, we have proved that any x clique inequality is implied by the constraint set of our formulation. The following theorem shows that there is no odd hole in the G_x graph.

Theorem 5.2.10 *There is no odd hole in graph G_x.*

PROOF. Assume there is a minimal odd hole H with $|H| \geq 5$. Let $H = x^1, x^2, ..., x^{|H|}, x^1$. Let c_{r_j} (for j in $\{1, 2, ..., |H|\}$) be the cores of the x^j and k_j be the sequence positions of x^j. Obviously, there are no three common cores among the c_{r_j}; otherwise, H would have at least one chord. Because $|H|$ is odd, there must be one j such that $r_j \leq r_{j+1} \leq r_{j+2}$. In this proof, if $j + d$ is greater than $|H|$, then it should be interpreted as $j + d - |H|$ $(d = 1, 2, ...)$; if $l - d$ is less than 0, then it should be interpreted as $l - d + |H|$ $(d = 1, 2, ...)$. We have the following two cases:

1. r_j, r_{j+1}, and r_{j+2} are all different. Based on Lemma 5.2.2, x^j conflicts with x^{j+2}. Thus (x^j, x^{j+2}) is an edge in G_x that forms a chord in H. So H is not a hole, which is a contradiction.

2. Two of r_j, r_{j+1}, and r_{j+2} are equal. Without loss of generality, assume r_j is less than r_{j+1} which is equal to r_{j+2}. Then $k_{j+1} < k_{j+2}$, otherwise, x^j would conflict with x^{j+2} based on Lemma 5.2.6. We now prove that for all l not equal to $j+1$, $j+2$, we have that r_l is less than r_{j+1}. For all l such that l and $l-1$ are not equal to $j+1$ or $j+2$, either $l-1 \neq j+3 \bmod |H|$ or $l \neq j$ holds; otherwise, $|H| \leq 4$. We have the following two cases:

   - If $l-1 \neq j+3$, then (x^l, x^{j+2}) and (x^{l-1}, x^{j+2}) cannot be the edges of G_x; otherwise H has chords. Since (x^l, x^{l-1}) is an edge of H and G_x (*i.e.*, x^l conflicts with x^{l-1}), based on Lemma 5.2.5, we have that either both r_l and r_{l-1} are greater than r_{j+2} or both r_l and r_{l-1} are less than r_{j+2}.
   - If $l \neq j$, then (x^l, x^{j+1}) and (x^{l-1}, x^{j+1}) cannot be the edges of G_x because H has no chord. Similarly, we have that either both r_l and r_{l-1} are greater than r_{j+1} or both r_l and r_{l-1} are less than r_{j+1}.

   Thus, for all l in $\{j, j-1, ..., j+4\}$, we have either both r_l and r_{l-1} are less than r_{j+1} and r_{j+2} or both r_l and r_{l-1} are greater than r_{j+1} and r_{j+2}. Because we have already assumed r_j less than r_{j+1}, for all l in $\{j, j-1, ..., j+4\}$, r_l is less than r_{j+1} and r_{j+2}. Since (x^{j+3}, x^{j+1}) is not an edge of G_x (otherwise H has chords), $k_{j+1} < k_{j+2}$ and $r_{j+3} < r_{j+1} = r_{j+2}$, based on Lemma 5.2.4, we have (x^{j+3}, x^{j+2}) is not an edge of G_x or H, which is a contradiction!

This completes the proof of this theorem.

□

Based on Lemma 5.2.8, we have the following lemma about a very simple y clique.

Lemma 5.2.11 *Any y-clique inequality consisting of only two y variables is implied by Constraints 4.2, 4.14, 4.15, 4.16 and 4.17.*

PROOF. Consider the x variables corresponding to the two y variables. There are at least two of these that are in conflict by Definitions 5.2.2 and 5.2.3. The sum of these two y variables are no more than that of these two x variables by Constraints 4.14 and 4.15. Lemma 5.2.8 completes the proof. □

Based on the results of Theorem 5.2.9, Theorem 5.2.10 and Lemma 5.2.11, we can see that our formulation captures the special structure of this problem so well that we do not have to explicitly add any x-clique inequality during the solving of our integer program. The experimental results show that most of the time, the optimal solution of our integer program's linear relaxation is integral. We do not need to consider more complex y-clique inequalities nor any further cuts.

5.2.2 Integrality of a Special Case

If the template contact graph does not contain the interaction preference edges, that is, it only contains the variable gap edges between two adjacent cores, then a simple dynamic programming algorithm can be used to solve the threading alignment problem [62] within time bounded by a low degree polynomial $O(Mn^2)$ where M is the number of template cores and n is the sequence length. In this subsection, we will show that in this case, the linear solution of the relaxed linear program is also integral.

Theorem 5.2.12 *If the interaction preferences are not considered, that is, the contact graph CMG only contains edges corresponding to the variable gaps, then the optimal linear solution of the relaxed linear program is integral.*

PROOF. Let $S^* = (x^*, y^*)$ denote the optimal linear solution. We will show that $S^* = (x^*, y^*)$ is a convex combination of feasible integral solutions if it is not integral. That is, there exists a set of K integral solutions $S_j = (x^j, y^j)$ and positive fractions α_j for j from 1 to K with $\sum_{i=1}^{K} \alpha_j = 1$, such that

$$S^* = \sum_{j=1}^{K} \alpha_j S^j \tag{5.1}$$

Let $(x^0, y^0) = (x^*, y^*)$. We have the following properties based on Constraint 4.14 and 4.15.

$$x^0_{i,l_i} = \sum_{l \in R[i,i+1,l_i]} y^0_{(i,l_i)(i+1,l)}, foralll_i \in D[i] \tag{5.2}$$

$$\sum_{l \in R[i+1,i,l_{i+1}]} y^0_{(i,l)(i+1,l_{i+1})} = x^0_{i+1,l_{i+1}}, forallll_{i+1} \in D[i+1] \quad (5.3)$$

$$\sum_{l \in D[i]} x^0_{i,l} = \sum_{k \in D[j]} x^0_{j,k}, forallli, j \quad (5.4)$$

We decompose (x^*, y^*) by repeating the following two steps until $(x^0, y^0) = 0$.

Step 1: Starting from core c_1, alternatively choose positive elements x^0_{1,l_1}, $y^0_{(1,l_1)(2,l_2)}$, x^0_{2,l_2}, $y^0_{(2,l_2)(3,l_3)}$, ..., $y^0_{(M-1,l_{M-1})(M,l_M)}$, x^0_{M,l_M} from vector (x^0, y^0) such that no conflict occurs among them. We can always do so. Given $x^0_{i,l_i} > 0$, $\sum_{l \in R[i,i+1,l_i]} y^0_{(i,l_i)(i+1,l)} = x^0_{i,l_i} > 0$, so we can choose one $y^0_{(i,l_i)(i+1,l_{i+1})} > 0$ with $l_{i+1} \in R[i, i+1, l_i]$. Given one $y^0_{(i,l_i)(i+1,l_{i+1})} > 0$, we can choose $x^0_{i+1,l_{i+1}}$ because $x^0_{i+1,l_{i+1}} \geq y^0_{(i,l_i)(i+1,l_{i+1})} > 0$. Obviously, we can always choose $x^0_{1,l_1} > 0$ because $\sum_{l \in D[1]} x^0_{1,l} > 0$. Let α_j denote the smallest element in the sequence x^0_{1,l_1}, $y^0_{(1,l_1)(2,l_2)}$, x^0_{2,l_2} ..., x^0_{M,l_M}.

Step 2: Let (x^j, y^j) denote a vector with $x^j_{1,l_1} = y^j_{(1,l_1)(2,l_2)} = x^j_{2,l_2} = ... = x^j_{M,l_M} = 1$ and other elements being 0. Based on the above choice, (x^j, y^j) is a feasible integral solution to the linear program. Let $(x^0, y^0) = (x^0, y^0) - \alpha_j(x^j, y^j)$. Obviously, $(x^0, y^0) \geq 0$. It is easy to verify that Equation 5.2, 5.3 and 5.4 still hold.

Assume that after K iterations, (x^0, y^0) becomes 0. Obviously, $\sum_{j=1}^{K} \alpha_j = \sum_{l \in D[1]} x^*_{1,l} = 1$ because at each iteration j, for any core c_i, $\sum_{l \in D[i]} x^0_{i,l}$ is decreased by α_j. Therefore, Equation 5.1 holds. However, for any linear program, an optimal solution cannot be a convex combination of other feasible solutions [21, 98]. Therefore, the optimal solution (x^*, y^*) is integral. □

The results of Theorem 5.2.9, Theorem 5.2.10 and Theorem 5.2.12 reveal that our formulation is very effective. In the next section, we will describe the behavior of our formulation on the actual protein data.

5.3 Experimental Results

In this section, we will investigate some experimental results on the integrality of the linear solutions of our integer program and the CPU time with respect to the sequence size.

5.3.1 Integrality of Linear Solutions

Table 5.1: Statistic of integrality of the linear solutions.

Test Set	Integrality Ratio	Maximu Branch Number	Sequence Fraction Average (Deviation)	Sequence Fraction Maximum	Template Faction Average (Deviation)	Template Fraction Maximum
Fischer	98.7	7	3.03 (3.04)	18	0.79 (1.04)	5
Holm	99.1	7	2.02 (1.94)	7	1.67 (1.91)	10
Lindhal	99.0	9	6.31 (5.72)	31	1.70 (1.88)	10

(1) *SFracNum(S)*: the number of threading pairs with the query sequence being S whose linear programs generate the fractional optimal solutions; (2) *TFracNum(T)*: the number of threading pairs with the template being T whose linear programs generate the fractional optimal solutions; (3)Integrality Ratio: percent of the linear solutions being integral; (4)Max Branch Number: the maximum number of branch nodes among all threading pairs; (5)Sequence Fraction Average: average of $SFracNum(S)$ among all sequences in the test set; (6)Sequence Fraction Deviation: standard deviation of $SFracNum(S)$ among all sequences in the test set; (7)Sequence Fraction Maximum: the maximum of $SFracNum(S)$ among all sequences; (8)Template Fraction Average: average of $TFracNum(T)$ among all templates in the test set; (9)Template Fraction Deviation: standard deviation of $TFracNum(T)$ among all templates in the test set; (10)Template Fraction Maximum: the maximum of $TFracNum(T)$ among all templates.

To evaluate the behaviors of the linear solutions of the relaxed linear programs, the sequences and templates from Fischer et al.'s test set [36], Lindahl and Elofsson's test set [75] and Holm and Sander's test set [48] are used. For Fischer et al.'s test set, all sequences are threaded against all templates in the test set. For Lindahl and Elofsson's and Holm and Sander's test sets, some sequences are randomly selected and threaded against all the templates in the corresponding test set. Those templates with no more than 4 cores or without interactions between cores are excluded from this experiment because their

corresponding linear programs always produce integral solutions. In our experiment, there are 68×262 threading pairs from Fischer et al.'s test set, 192×712 pairs from Lindahl and Elofsson's test set, and 197×238 pairs from Holm and Sander's test set. For each threading pair, we record whether the optimal linear solution of its linear program is integral and the number of branch nodes if the solution is fractional.

Let *SFracNum(S)* denote the number of threading pairs with the query sequence being S whose linear programs generate the fractional optimal solutions, and *TFracNum(T)* denote the number of threading pairs with the template being T whose linear programs generate the fractional optimal solutions. We calculate some indices as shown in Table 5.1. Approximately 99% of linear programs generate integral solutions directly. This validates the influence of the theoretical results proved in Section 5.2. Since linear programs can be solved in polynomial time, in a practical sense, threading alignment problems can be solved within polynomial time with our scoring function and alignment model. Based on the mean and standard deviation statistics in Table 5.1, it seems unlikely that there are some specific sequences or templates which can lead to a much greater chance of fractional linear solutions.

Table 5.2: Correlation coefficient between the number of fractional linear solutions and sequence length, and four template structure features: topological complexity, #cores (number of cores), template size and #edges (number of edges in the structure contact graph).

	topological complexity	#cores	template size	#edges	sequence length
Holm	0.0389	0.0879	-0.0742	0.1155	0.1467
Lindhal	0.0774	0.0587	-0.1510	0.0816	-0.0506
Fischer	-0.1107	0.2207	0.1230	0.1867	0.1991

Table 5.2 shows the correlation coefficients between *TFracNum(T)* and the structure features of the templates, and the correlation coefficients between *SFracNum(S)* and the sequence size. As shown in these two tables, there is a very weak relationship between the non-integrality of the linear solutions and the topological features of the templates

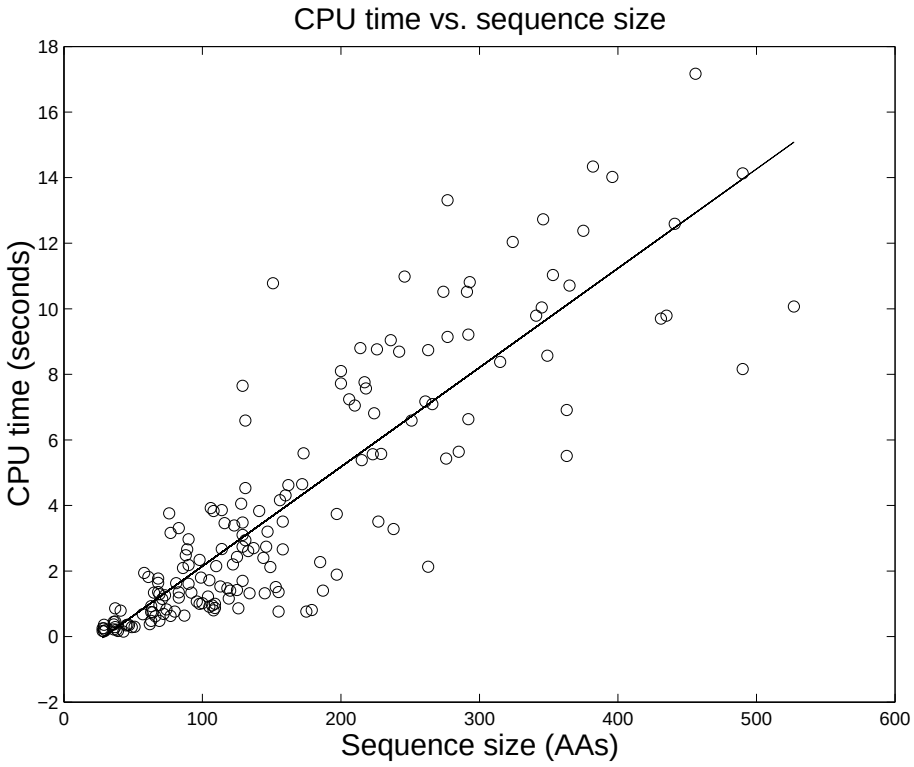

Figure 5.1: CPU time of threading 170 sequences to template 119l.

and sequences. This means that for any linear program derived from any threading pair with any topology, there appears to be a good chance of generating an integral optimal solution directly. Our integer programming approach is effective and efficient for most of the threading pairs.

5.3.2 CPU time

Besides the previous feature of our algorithm, another advantage is that the computing time increases roughly with the sequence size. Figure 5.1 reflects the CPU time of aligning a template 119l with length 162 to 170 sequences (chosen randomly from Lindahl and Elofsson's benchmark) with sizes ranging from 28 to 527. The figure shows that the computing time of our algorithm increases very slowly (almost linearly rather than exponentially) with

respect to the sequence size.

Figure 5.2 shows the CPU time on the supercomputer FLEXOR with 20G memory and 40 CPUs of 400MHz for the prediction of each CAFASP3 target sequence which will be described in Chapter 8. There are in total 62 targets and 3236 protein templates in our template database. As shown in this figure, the CPU time increases very slowly with respect to the sequence size except that it takes about 45 hours for one target(t0174). After carefully examining the running time of threading t0174 to each template, we discovered that there are 30 templates for which it takes about 15 hours to thread t0174 to them. The reason is that we have used the predicted secondary structure of the target sequence to exclude some alignment positions of each template core before calling the linear programming package. The predicted secondary structure of t0174 is very bad, thus very few alignment positions are excluded for each core, which leads to a linear program with more than ten millions of variables for these 30 templates.

5.4 Discussion

Our theoretical analysis cannot strictly prove that our formulation of the protein threading problem is very strong, because it is impossible to mathematically define "strong", but it at least can provide us with some intuition that our formulation is very effective. The experimental results on integrality of the linear solutions suggest that, in practice, this version of the protein threading problem can be solved by a linear programming approach. In particular, the NP-hardness of the protein threading problem is only theoretically meaningful. There appears to be very little chance that the adverse instances constructed in proving the NP-hardness result exist in the real world.

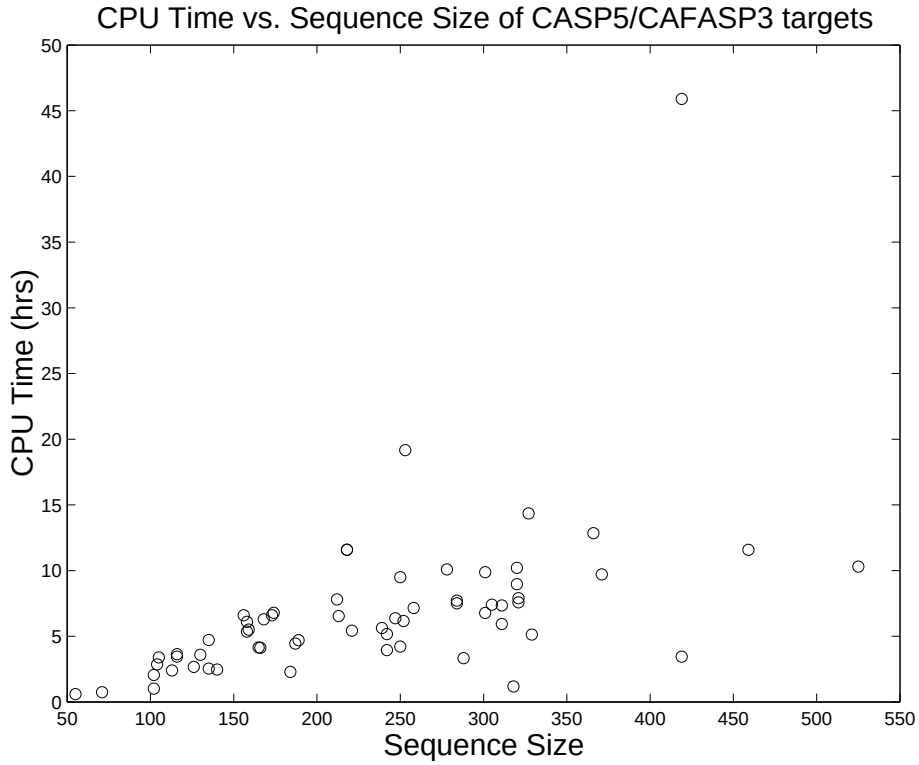

Figure 5.2: CPU time of threading 62 CAFASP3 target sequences to 3236 templates.

Chapter 6

Improving Efficiency by Graph Reduction

In the previous chapters, we have formulated the protein threading problem as a large scale linear programming (LP) problem. In order to further improve the computational efficiency of our LP approach, this chapter proposes a graph reduction technique to reduce a template contact graph to a new one with fewer vertices and edges [125]. Our graph reduction operation is formalized to minimize the number of variables, constraints, and non-zero elements in the constraint matrix of our linear programs. These three factors are key to the computational time of solving linear programs by both the Simplex method and the interior-point method. The experiments show that the more a template contact graph can be reduced, the more computational efficiency improvement can be attained and that on average, the computational efficiency of threading any long sequence to the whole template database can be improved by 30%. A generalization of our graph reduction technique and our LP formulation is also discussed in this chapter.

6.1 LP Approach vs. Divide-and-Conquer Method

Based on the observation that many template contact graphs are "regular", PROSPECT [132] uses a divide-and-conquer algorithm to search for the globally optimal alignment between the target sequence and each template. The idea is to split a template into two

subsegments such that each subsegment is connected to as few external vertices (i.e., the cores not in this subsegment) as possible, recursively align each subsegment to the target sequence, and finally merge the alignments of two subsegments to form an alignment for the whole segment. The challenge is how to split the template to render the number of external vertices, which the alignment of each subsegment to the target sequence depends on, as small as possible. The computational complexity is dominated by the alignment of one subsegment with the maximum number of external vertices. The best splitting method will minimize the maximum number of external vertices of all the generated subsegments. PROSPECT uses a measure of topological complexity (TC) to describe this value. The value of TC can be regarded as the quantification of the "regularity" of contact graphs.

In this section, we outline the divide-and-conquer algorithm to align one segment to a target sequence. For a more detailed exposition, please refer to Xu et al's paper [132]. Our description is slightly different. Given a template contact graph CMG, for any two cores c_i and c_j $(i < j)$, let $In(c_i, c_j)$ denote $\{c_i, c_{i+1}, ..., c_j\}$ and $N(c_i, c_j) = V(CMG) - In(c_i, c_j)$. Let $E(c_i, c_j)$ denote the set of all the edges with both ends in set $In(c_i, c_j)$ and $XE(c_i, c_j)$ the set of all edges with one end in $In(c_i, c_j)$ and the other end in $N(c_i, c_j)$.

Definition 6.1.1 *Given a segment (c_i, c_j), a vertex cover of edge set $XE(c_i, c_j)$ is called an anchor set of this segment.*

Given a segment (c_i, c_j), we split it into two subsegments (c_i, c_k) and (c_{k+1}, c_j) at position k. Let $Anchor(c_i, c_j)$ denote one anchor set of segment (c_i, c_j). Let $CUT(k)$ be those edges with one end in each subsegment and $CutCover(k)$ be a set of vertices which covers all the edges in $CUT(k)$. The cut at position k also splits $Anchor(c_i, c_j)$ into two disjoint subsets $X^1(k)$ and $X^2(k)$, where $X^1(k)$ contains the cores in $\{c_i, c_{i+1}, ..., c_k\}$ or adjacent to the cores in $\{c_i, c_{i+1}, ..., c_k\}$ and $X^2(k)$ contains the cores in $\{c_{k+1}, ..., c_j\}$ or adjacent to the cores in $\{c_{k+1}, ..., c_j\}$. Obviously, $X^1(k) \cup CutCover(k)$ and $X^2(k) \cup CutCover(k)$ are anchor sets of segment (c_i, c_k) and segment (c_{k+1}, c_j) respectively. Let $SCORE((c_i, c_j), Anchor(c_i, c_j))$ denote the optimal alignment score of aligning the sequence to segment (c_i, c_j) given the alignment positions of all cores in $Anchor(c_i, c_j)$. Let $D(C)$ denote all feasible combinations of alignment positions of vertices in set C, where C is a subset of $\{c_1, c_2, ..., c_M\}$. If we divide the segment (c_i, c_j) at position k, then we have the following equation:

$$
\begin{aligned}
SCORE((c_i, c_j), Anchor(c_i, c_j)) &= \min_{D(CutCover(k))} \{SCORE((c_i, c_k), X^1(k) \cup CutCover(k)) \\
&+ SCORE((c_{k+1}, c_j), X^2(k) \cup CutCover(k))\} \qquad (6.1)
\end{aligned}
$$

In Equation 6.1, we enumerate all possible alignment positions of *CutCover(k)* and find the one that minimizes the right hand side of this equation. Then the optimal alignment between a template and a sequence can be calculated by calling $SCORE((c_1, c_M), \phi)$. Now consider the computational complexity of the recursive algorithm based on Equation 6.1. Let $T((c_i, c_j), Anchor(c_i, c_j))$ denote the computational time of $SCORE((c_i, c_j), Anchor(c_i, c_j))$, then we have

$$
\begin{aligned}
T((c_i, c_j), Anchor(c_i, c_j)) &= T((c_i, c_k), X^1(k) \cup CutCover(k)) \\
&+ T((c_{k+1}, c_j), X^2(k) \cup CutCover(k)) \\
&+ O(n^{(|CutCover(k)|+|Anchor(c_i,c_j)|)}) \qquad (6.2)
\end{aligned}
$$

where the third item of the right hand side of Equation 6.2 is the computational time needed for merging the two subsegments. Given a segment and its anchor set, in order to minimize its alignment computational time, we need to choose a proper cut position k, and a proper cut cover $CutCover(k)$ to minimize the maximum of the three items in the right hand side of Equation 6.2 during the recursive process. In their paper [132], Xu et al. have proposed an algorithm MIN_MAX_CUT to search for the optimal partition scheme of the whole template and the optimal set cover of cut edges at each cut position. The topological complexity of a template contact graph TC is defined as the maximum among all $|X^1(k) \cup CutCover(k)|$ and $|X^2(k) \cup CutCover(k)|$. MIN_MAX_CUT is designed to minimize TC.

The computational complexity of the divide-and-conquer method is $O(Mn^{\frac{3}{2}TC+1})$ and the memory usage is $O(Mn^{TC+1})$ where n is the sequence size and M the number of template cores. Therefore, PROSPECT is efficient for only those templates with a contact graph having a low topological complexity. PROSPECT has difficulty in both memory and CPU usage when threading a long sequence to a template with a topologically complex contact graph ($TC \geq 4$). If the contact distance cutoff is $7\dot{A}$, approximately 25% of the

templates have a contact graph with $TC \geq 4$. Nonetheless, these templates dominate the computational time of threading one sequence to the entire template database.

As reported in the previous chapters, for almost all threading instances, our LP formulation produces integral solutions directly. Therefore, we can use the computational complexity of solving a LP to approximate the computational time of our integer program. A linear program can be solved within $O(\hat{n}^3 L)$ by the interior-point method where $\hat{n}$ is the number of variables and L is the input size[1] [112]. Our LP formulation has $O(|E|n^2)$ variables and $O(|E|n^2)$ non-zero elements in the constraint matrix where $E = E(CMG)$. As such, the computational complexity of our LP is $O(|E|^3 n^6 L)$ if the interior-point method is used. Another advantage of our LP approach is that if the Simplex method is used, the memory used is roughly $O(|E|n^2)$, the number of active elements in the constraint matrix. Therefore, our LP approach can deal very well with those templates with a "non-regular" contact graph. However, $O(|E|^3 n^6 L)$ is expensive for those templates with a very "regular" contact graph. A straightforward idea is to combine our LP approach and the divide-and-conquer method. For those templates with "regular" contact graphs, we use the divide-and-conquer method, whereas for those with "non-regular" contact graphs, we use the LP approach instead.

Can we go beyond such a simple idea? Yes. Further observation shows that even a template with a topologically complex contact graph often contains some subsegments inducing topologically simple subgraphs. We can exploit this kind of partial regularity by employing a graph reduction technique which can reduce such a subsegment into a single edge. The subsegment can be aligned to the sequence by the divide-and-conquer method because of the low topological complexity of its contact graph. The original contact graph is reduced to a new "non-regular" contact graph with fewer vertices and edges. Our LP approach can be used to align the template to the sequence, based on the reduced contact graph. The resultant LP formulation contains fewer variables, constraints and non-zero elements in the constraint matrix. If the cost of aligning the subsegments to the sequence is relatively small, then the computational efficiency of our LP approach can be improved due to the smaller number of variables, constraints and non-zero elements in the constraint matrix. In the next section, we will describe in detail how to formalize this idea as a

[1] In most of our cases, the Simplex method is faster than the interior-point method.

graph reduction problem which enables us to combine the LP approach and the divide-and-conquer method to speed up our protein threading algorithm.

6.2 Contact Graph Reduction

For the purpose of simplicity, we provide the following definition.

Definition 6.2.1 *Given a template contact graph CMG and a segment (c_i, c_j), a segment subgraph $SG(c_i, c_j) = (V(SG), E(SG))$ derived from segment $((c_i, c_j))$ is an induced subgraph on $V(SG)$, where $V(SG) = \{c_i, c_{i+1}, ..., c_j\}$.*

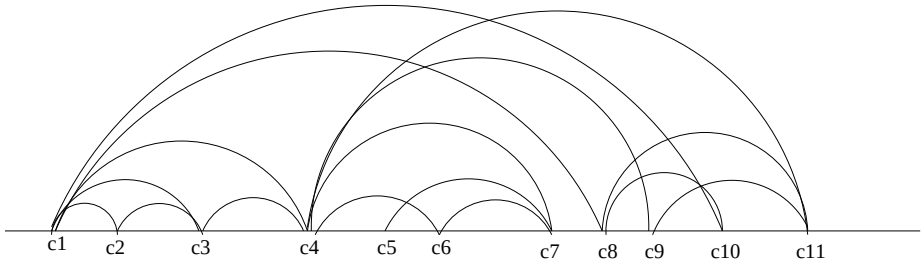

Figure 6.1: Template contact graph.

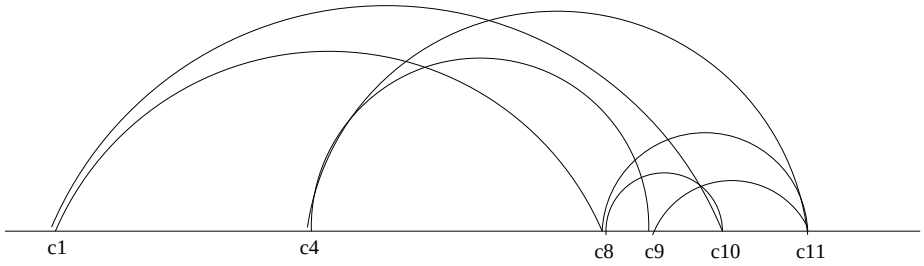

Figure 6.2: Reduced template contact graph.

Let $N(c_i)$ denote the set of vertices (cores) adjacent to core c_i in the template contact graph. If there are two cores $c_i, c_k (i < k)$ such that for all j, $i < j < k$, $N(c_j) \in$

$\{c_i, c_{i+1}, ..., c_k\}$, and the segment subgraph $SG(c_i, c_k)$ has a low topological complexity, then we can first align segment (c_i,c_k) to the sequence by the divide-and-conquer method within time bounded by low-degree polynomial and calculate the new pairwise scores $\hat{P}(c_i, c_k, l_i, l_k)$ for all $l_i \in D[i]$ and $l_k \in D[k]$. Given a pair l_i, l_k, the optimal alignment positions of $c_{i+1}, ..., c_{k-1}$ are completely determined by this process because they have no interaction relationship with the cores not within segment (c_i, c_k). Therefore, segment (c_i, c_k) can be reduced to a single edge and the pairwise scores related to it are $\hat{P}(c_i, c_k, l_i, l_k)$. The original contact graph is reduced to a smaller graph. As shown in Figure 6.1 and Figure 6.2, segment (c_1, c_4) and segment (c_4, c_8) are reduced to two edges respectively. This occurs because the segment subgraphs $SG(c_1, c_4)$ and $SG(c_4, c_8)$ have a low topological complexity. Both segments can be aligned to the target sequence by the divide-and-conquer method within $O(L_{seg}n^3)$ and $O(L_{seg}n^4)$ where L_{seg} is the length of the segment [132].

Formally, we define this kind of graph reduction as follows.

Definition 6.2.2 *Given a template contact graph $CMG = (V, E)$, $|V| \geq 2$, a subset $RV = \{c_{i_1}, c_{i_2}, ..., c_{i_k}\}$ ($i_1 < i_2, ..., < i_k$, $k \geq 2$) of V is called a* Reduced Vertex Set *if for all $e = (c_l, c_p) \in E$, either $c_l, c_p \in RV$ or there exists a unique j ($1 \leq j \leq k$) such that $l, p \in \{i_j, i_j + 1, ..., i_{j+1}\}$.* [2]

Definition 6.2.3 *Given a template contact graph $CMG = (V(CMG), E(CMG))$, and its* Reduced Vertex Set *RV, we construct its* Reduced Contact Graph *$RCMG$ according to the following procedures:*

1. *Let $V(RCMG)$ be RV;*
2. *For all $v_1, v_2 \in V(RCMG)$, if $(v_1, v_2) \in E(CMG)$, then add (v_1, v_2) to $E(RCMG)$;*
3. *For any segment $(c_{i_j}, c_{i_{j+1}})$, add the edge $(c_{i_j}, c_{i_{j+1}})$ to $E(RCMG)$, if it does not exist yet.*

The cost of constructing a reduced contact graph is $O(M^2)$. Based on the above definition, the reduced contact graph is still a contact graph. Therefore, our LP approach to protein threading can be used on the reduced contact graph. According to Definition

[2] if j equals k, then $j + 1$ should be interpreted as 1, that is, $\{i_j, i_j + 1, ..., i_{j+1}\} = \{i_j, ..., M, 1, ..., i_1\}$.

6.2.2 and Definition 6.2.3, we can see that for any given template contact graph, we may be able to induce several different reduced contact graphs from it. Which one is the best in terms of computational complexity? We first make the following definition to measure the extent to which a graph is reduced.

Definition 6.2.4 *The edge reduction ratio of a contact graph is defined as the ratio between the number of edges in the reduced contact graph and the number of edges in the original contact graph.*

According to Definition 6.2.4, the smaller the edge reduction ratio, the more the contact graph is reduced.

Let $RV = \{c_{i_1}, c_{i_2}, ..., c_{i_p}\}$ denote the reduced vertex set of the contact graph after the graph reduction operation. Let T_{seg} denote the computational complexity of threading the target sequence to all segments $(c_{i_r}, c_{i_{r+1}})$ $(r = 1, 2, .., p)$ by the divide-and-conquer method and $T_{re}(LP)$ that of threading the sequence to the template with the reduced contact graph by our LP approach. Ideally, the best graph reduction operation should minimize the sum of T_{seg} and $T_{re}(LP)$. If we only consider template contact graphs with a high topological complexity (i.e., $TC \geq 4$), then our objective is to minimize $T_{re}(LP)$ such that $T_{seg} = o(T_{re}(LP))$ because the LP approach is more efficient than the divide-and-conquer method on these templates. As mentioned before, our LP formulation has $O(|E|n^2)$ variables, $O(Mn)$ constraints, and $O(|E|n^2)$ non-zero elements in the constraint matrix. In order to minimize $T_{re}(LP)$, we should make the number of variables, constraints and non-zero elements as small as possible. Therefore, our graph reduction problem can be formalized as follows.

Given a template contact graph, find an optimal reduced vertex set to minimize its edge graph ratio such that $T_{seg} = o(T_{re}(LP))$.

To summarize, our improved LP approach consists of the following three major steps:

- Find the optimal reduced vertex set $RV = \{c_{i_1}, c_{i_2}, ..., c_{i_k}\}$ by the algorithm discussed in the next section;
- Align each subsegment $(c_{i_j}, c_{i_{j+1}})$ to the sequence by the divide-and-conquer method;

- Employ our LP approach to align the template to the target sequence based on the reduced contact graph.

An interesting question is that if T_{seg} is bounded by a low-degree polynomial time, is there a limit to the edge reduction ratio for all (mathematically) possible contact graphs? Lemma 6.2.1, 6.2.3 and 6.2.4 give some theoretical results about the edge reduction ratio.

Lemma 6.2.1 *Given a template with a contact graph having only $\frac{M}{2}$ arc edges $(i, i+\frac{M}{2})$ $(i = 1, 2, ..., \frac{M}{2})$, the computational complexity of its optimal alignment to a sequence of length n by the divide-and-conquer method is $\Omega(n^{\frac{M}{2}})$.*

PROOF. For any partition scheme adopted by the divide-and-conquer method, we can always find a segment (c_i, c_j) of length no less than $\frac{M}{2}$ such that its alignment to the sequence is constructed from the alignments of two subsegments (c_i, c_k) and (c_{k+1}, c_j), each of which has length less than $\frac{M}{2}$. There are in total $\frac{M}{2}$ arc edges with at least one end in segment (c_i, c_j). These arc edges can be grouped into two disjoint subsets A and B. A is the set of arc edges with only one end in this segment and B the set of arc edges with both ends in this segment. Obviously, any arc edge in B has one end in segment (c_i, c_k) and the other end in segment (c_{k+1}, c_j). Therefore, in merging the alignments of these two subsegments, the divide-and-conquer method has to enumerate (i) the alignment positions of at least one end of each arc edge in B; and (ii) the alignment positions of at least one end of each arc edge in A. Because any two arcs in set $A \cup B$ are not incident to the same vertex, the algorithm has to enumerate in total $O(n^{|A|+|B|})$ possible combinations of alignment positions in order to merge these two subsegments into segment (c_i, c_j). Therefore, the computational complexity of aligning this template to the sequence of length n is $\Omega(n^{\frac{M}{2}})$. □

Lemma 6.2.1 gives an extreme case of a "non-regular" contact graph. The divide-and-conquer algorithm will encounter difficulty in aligning the template to the sequence with such a sparse contact graph. This is why we say the divide-and-conquer method is good for templates with "regular" contact graphs rather than with sparse contact graphs. Our LP approach for such a template is very efficient because the number of variables, constraints and non-zero elements in the constraint matrix is relatively small. Should every contact

graph be reduced to such a case or close to one, the computational efficiency of our LP approach would be improved greatly.

Definition 6.2.5 *A template contact graph is called a nested contact graph if it contains no more than three vertices or for any two different arc edges* (i_1, j_1) *and* (i_2, j_2)*, either* $i_1 \leq i_2 < j_2 \leq j_1$ *or* $i_2 \leq i_1 < j_1 \leq j_2$*.*

Lemma 6.2.2 *There exists a nested contact graph with* M *vertices and* $M - 2$ *arc edges.*

PROOF. Obvious. □

Lemma 6.2.3 *The optimal alignment between a sequence of length* n *and a template with a nested contact graph can be solved by the divide-and-conquer method within* $O(Mn^3)$ *time.*

PROOF. For a segment subgraph $SG(c_i, c_j)$, if it is a nested contact graph, then we can always find a core c_k $(i < k < j)$ such that the two segment subgraphs $SG(c_i, c_k)$ and $SG(c_{k+1}, c_j)$ are nested contact graphs. Divide segment (c_i, c_j) at position k. The computational complexity of merging these two subsegments is $O(n^3)$, by enumerating the alignment positions of c_i, c_k and c_j. Simple induction shows that the alignment of this template to any sequence can be solved within $O(Mn^3)$ time. □

The following lemma shows that theoretically, there are some templates which are extremely inefficient to be threaded by the divide-and-conquer method, but have a template contact graph which can be reduced greatly.

Lemma 6.2.4 *Given a ratio* $\epsilon > 0$ *and an integer* $m > 0$*, theoretically, there exists a template such that (i) the computational complexity of its alignment to a sequence of length* n *by the divide-and-conquer method is* $\Omega(n^m)$*; (ii) its contact graph can be reduced such that its edge reduction ratio is no more than* ϵ *and the computational complexity of aligning the reduced segments to the sequence by the divide-and-conquer method is* $O(L_{seg}n^3)$ *where* L_{seg} *is the total length of all the reduced segments.*

PROOF. Construct a contact graph G_1 with $2m$ vertices and m arc edges $(i, i+m)$ $(i = 1, 2, ..., m)$. Then construct a new graph G by expanding each gap edge in G_1 into a nested contact graph with d vertices and $d-2$ edges. The computational complexity of aligning the template with contact graph G to the sequence by the divide-and-conquer method is $\Omega(n^m)$ based on Lemma 6.2.1. G can be reduced into graph G_1 by selecting the reduced vertex set as $V(G_1)$. The computational complexity of aligning the reduced segments to the sequence is $O(L_{seg}n^3)$. A proper choice of d can make the edge reduction ratio smaller than ϵ. □

Based on Lemma 6.2.4, we can see that mathematically, even if T_{seg} is $O(L_{seg}n^3)$, there still exist some contact graphs such that the graph reduction technique can make their edge reduction ratios very small, and as such, improve the computational efficiency of our LP approach greatly.

The remaining problem is to design a graph reduction algorithm to minimize the edge reduction ratio given $T_{seg} = O(L_{seg}n^k)$, $k \geq 3$, which will be described in the next section.

6.3 Graph Reduction Algorithm

In this section, we will discuss the algorithm for the graph reduction operation. Given a template contact graph CMG and a small constant k $(k \geq 3)$, our goal is to find an optimal reduced vertex set RV such that the edge reduction ratio is minimized and $T_{seg} = O(L_{seg}n^k)$ where L_{seg} is the number of cores of all reduced segments. This is achieved by Algorithm 6.3 below.

Notice that for each template, we only need to search for its optimal reduced vertex set once, given a T_{seg}, regardless of how many sequences being threaded to it; therefore, we do not care too much about the computational efficiency of our graph reduction algorithm. Our graph reduction algorithm is straightforward but far from efficient. A simple pruning strategy can improve its efficiency greatly. Let MIN_MAX_CUT denote the algorithm to calculate the computational complexity of threading a target sequence to a template segment by the divide-and-conquer method. MIN_NAX_CUT is discussed in detail in Xu et al.'s paper [132].

Algorithm 6.3 enumerates all the feasible reduced vertex sets, and then calculates

the computational complexity of aligning each segment to the sequence by calling the MIN_MAX_CUT algorithm. Secondly, Algorithm 6.3 checks whether $T_{seg} = O(L_{seg}n^k)$ and whether the edge reduction ratio size is minimal or not.

REDUCE-k(CMG, RV, k)

1: $RV \leftarrow V(CMG)$
2: $EdgeReductionRatio \leftarrow 1$
3: **for all** feasible reduced vertex sets rv **do**
4: $T_{seg} \leftarrow 0$
5: **for all** segments $(c_{i_r}, c_{i_{r+1}})$ with $i_{r+1} - i_r \geq 2$ **do**
6: $T_{seg} \leftarrow T_{seg} + MIN\_MAX\_CUT(c_{i_r}, c_{i_{r+1}})$ {Calculate the computational complexity of threading a sequence to all segments}
7: **end for**
8: **if** $T_{seg} \leq 2L_{seg}n^k$ **then**
9: calculate new edge reduction ratio ERR
10: **if** $EdgeReductionRatio > ERR$ **then**
11: $EdgeReductionRatio \leftarrow ERR$
12: $RV \leftarrow rv$
13: **end if**
14: **end if**
15: **end for**

Algorithm 1: Calculate RV subject to $T_{seg} = O(L_{seg}n^k)$, $k \geq 3$

6.4 Experimental Results

In order to test the effectiveness of our graph reduction operation, the efficiency of our original LP algorithm is compared with the improved algorithm proposed in this chapter by threading approximately 60 sequences to all the templates in our template database. The 60 sequences are chosen from the CASP5 target sequences [83], the LiveBench target sequences [95] and the sequences submitted by Genesilico group (http://www.genesilico.pl)

to the RAPTOR server, with sizes ranging from 330 to 450 residues. We do not experiment on short sequences because RAPTOR runs faster on them directly [3]. We run our test on Flexor at the University of Waterloo, which is a Silicon Graphics Origin 3800 system, with 40 400 MHz MIPS R12000 CPUs and 20 GB of RAM. For the sake of quantified comparison of the two algorithms, we define *CPU time ratio* as follows.

Definition 6.4.1 *For a template, CPU time ratio refers to the ratio between the total CPU time of our improved LP approach for threading all test sequences to this template and that of our original LP algorithm. For a sequence, CPU time ratio refers to the ratio between the total CPU time of our improved LP approach for threading this sequence to the entire template database* [4] *and that of our original LP algorithm.*

In our experiments, all the template contact graphs are reduced to minimal edge reduction ratio such that $T_{seg} = O(L_{seg}n^4)$. This graph reduction operation gives a best overall performance. First we examine the relationship between the computational efficiency improvement and the contact graph edge reduction ratio. As shown in Figure 6.3, the CPU time ratio is approximately the square of the edge reduction ratio, which means a reduction of several edges can greatly improve the computational efficiency of our LP approach. For almost all templates whose contact graphs are reduced, the computational efficiency is improved. For the templates with an irreducible contact graph, their CPU time ratios ranges from 95% to 105%. Therefore, the measurement error in our experiment is approximately 5%. All the templates used for this figure have a contact graph with topological complexity no less than 4.

Next we examine the relationship between the computational efficiency improvement and the target sequence size. As shown in Figure 6.4, the computational efficiency of threading each of the sequences is improved to some extent. The CPU time ratio ranges from 50% to 80%, that is, the computational efficiency is improved by 20% to 50%. The average CPU time ratio is 70% and its standard deviation is 5%. This indicates that the graph reduction operation can improve the computational efficiency by 30% in average for threading a sequence to the entire template database.

[3] The average size of protein sequences is $250 \sim 300$ AAs.

[4] Our template database contains about 3200 templates.

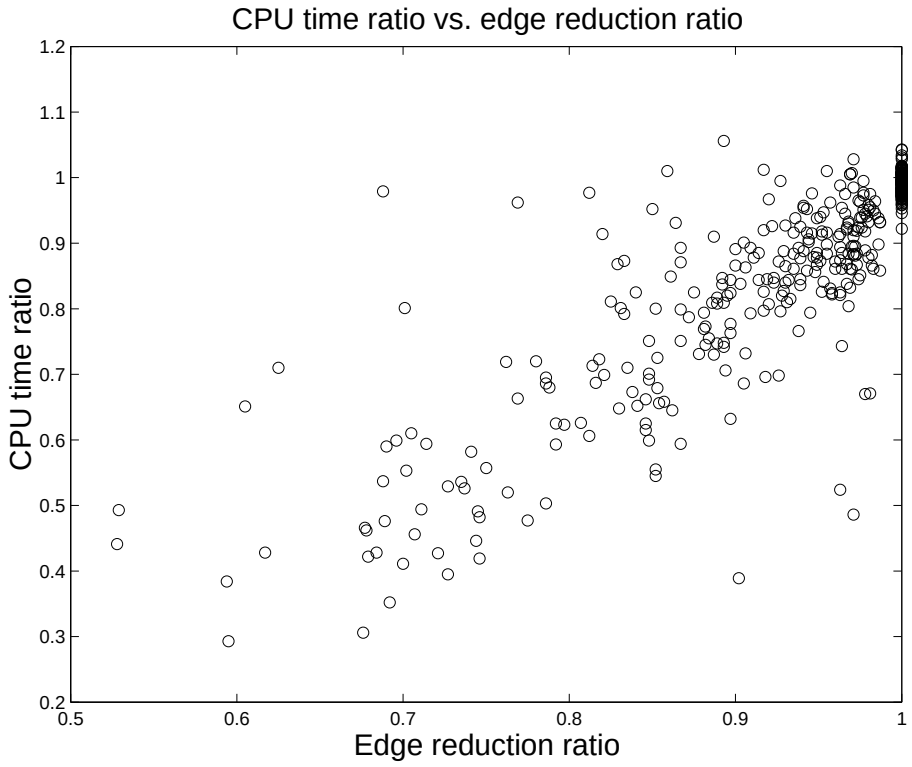

Figure 6.3: Relationship between the computational efficiency improvement and the edge reduction ratio.

6.5 Generalization

In this section, we will present a hypergraph-based integer program formulation which can be regarded as the generalization of a simple contact graph-based integer program formulation. This kind of formulation can be used for two aspects. First, if the threading scoring function takes into account the contact potential among multiple residues rather than just two residues, then we need to model the contact map of a template as a hypergraph, that is, we need to deal with a hypergraph-based threading problem directly. Secondly, even if we consider only the pairwise contact potential and generalize the definitions of reduced vertex set (Definition 6.2.2) and reduced contact graph (Definition 6.2.3) to Definition 6.5.1 and Definition 6.5.2 respectively, then after the graph reduction operation, the reduced

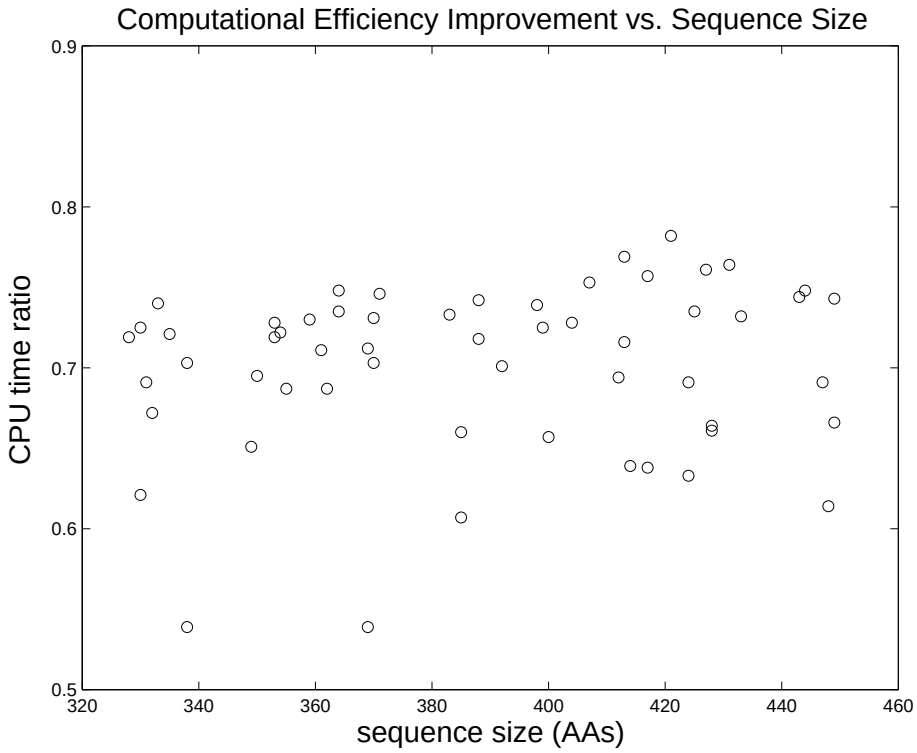

Figure 6.4: Relationship between the computational efficiency improvement and the sequence size.

contact graph can become a hypergraph. In order to employ the LP approach, we need to generalize our simple graph-based integer program formulation to the hypergraph-based formulation.

Definition 6.5.1 *Given a template contact graph* $CMG = (V, E)$, $|V| \geq 2$, *a subset* $RV = \{c_{i_1}, c_{i_2}, ..., c_{i_k}\}$ $(i_1 < i_2, ..., < i_k, k \geq 2)$ *of* V *is called a* Reduced Vertex Set *if* $\forall e = (c_l, c_p) \in E$ $(l < p)$, *then either at least one of* c_l, c_p *is in* RV *or there exists a unique* j $(1 \leq j \leq k)$ *such that* $i_j < l < p < i_{j+1}$.

Definition 6.5.2 *Given a template contact graph* $CMG = (V(CMG), E(CMG))$, *and its* Reduced Vertex Set RV, *its* Reduced Contact Graph $RCMG$ *can be constructed according*

to the following procedures:

1. *Let $V(RCMG)$ be RV;*
2. *For all $v_1, v_2 \in V(RCMG)$, if $(v_1, v_2) \in E(CMG)$, then add (v_1, v_2) to $E(RCMG)$;*
3. *For any segment $(c_{i_j}, c_{i_{j+1}})$, add the edge $(c_{i_j}, c_{i_{j+1}})$ to $E(RCMG)$ if it does not exist yet.*
4. *For any segment $(c_{i_r}, c_{i_{r+1}})$, let $c_{i_{j_1}}$,$c_{i_{j_2}}$,..., $c_{i_{j_p}}$ denote all cores in RV which are adjacent to at least one core within segment $(c_{i_r}, c_{i_{r+1}})$. We add one hyperedge $(c_{i_r}, c_{i_{r+1}}, c_{i_{j_1}}, c_{i_{j_2}}, ..., c_{i_{j_p}})$ to $E(RCMG)$.*

We also need to generalize the definition of contact map graph to the following.

Definition 6.5.3 *We use an undirected contact hypergraph $HCMG = (V, HE)$ to model the contact map graph of a protein template structure , $V = \{c_1, c_2, ..., c_M\}$, where c_i denotes the i^{th} core and $HE = \{e = (c_{i_1}, c_{i_2}, ..., c_{i_k}) |$ there is one multiple-body interaction among $c_{i_1}, c_{i_2}, ..., c_{i_k}\}$.*

The only difference between the hypergraph-based threading problem and the simple graph-based threading problem is that the scoring function of the hypergraph-based threading consists of the multiple-body interaction preferences. For any $e \in HE$,$e = (c_{i_1}, c_{i_2}, ..., c_{i_k})$, let $P(l_1, l_2, ..., l_k, e)$ denote the multiple-body interaction score when c_{i_j} is aligned to sequence position l_j $(1 \leq i \leq k)$. The objective function of hypergraph-based threading is to minimize $\sum_{c_i \in V(HCMG)} F(c_i, l_i) + \sum_{e \in HE} P(l_1, l_2, ..., l_k, e)$. Let x_{i,l_i} be a 0-1 variable indicating if core c_i is aligned to sequence position l_i. For any $e = (c_{i_1}, c_{i_2}, ..., c_{i_k}) \in HE$, let $y_{(e,l_1,l_2,...,l_k)}$ be a 0-1 variable indicating the multiple-body interaction among x variable x_{i_1,l_1}, x_{i_2,l_2}, ..., x_{i_k,l_k}, that is, $y_{(e,l_1,l_2,...,l_k)} = 1$ if and only if core c_{i_r} is aligned to sequence position l_r $(r = 1, 2, ..., k)$ simultaneously.

We could formulate the objective function of the hypergraph based threading as follows:

$$\min \sum x_{i_r,l_r} F(l_r, i_r) + \sum y_{(e,l_1,l_2,...,l_k)} P_{l_1,l_2,...,l_k,e}$$

Where $F(l_r, i_r)$ denotes the singleton score such as the fitness score and the mutation score when core i_r is aligned to sequence position l_r, and $P(l_1, l_2, ..., l_k, e)$ the multiple-body interaction score when $e = (c_{i_1}, c_{i_2}, ..., c_{i_k})$ and core c_{i_r} is aligned to sequence position l_r.

The constraint set can be formulated as follows:

$$\begin{aligned}
\sum_{l \in D[i]} x_{i,l} &= 1, i = 1, 2, ..., M; \\
x_{i_r,l_r} &= \sum_{l_p, p \neq r} y_{(e,l_1,l_2,...,l_k)}, \forall e = (c_{i_1}, c_{i_2}, ..., c_{i_k}) \in HE, \forall l_r \in D[i_r]; \\
y_{(e,l_1,l_2,...,l_k)} &\in \{0,1\}, e \in HE; \\
x_{i,l_i} &\in \{0,1\}, l_i \in D[i], i = 1, 2, ..., M.
\end{aligned}$$

We have conducted some preliminary experiments on reducing a simple contact graph to a hypergraph. Our experimental results demonstrate that there is no computational efficiency improvement. Conversely, the computational efficiency is unacceptable if the number of vertices contained in one hyperedge is more than three. The key factor is that the number of variables involved in the formulation is very large, if a long sequence being threaded. This suggests that it is computationally expensive to optimize the scoring function that incorporates multiple-body interactions.

6.6 Conclusion

In this chapter, we have discussed how to reduce template contact graphs to minimize the number of variables, constraints and non-zero elements contained in our linear programs that formulate the protein threading problem. This technique can improve the computational efficiency by about 30% for threading any long sequence to the entire template database. The computational efficiency improvement is highly related to how much the template contact graph can be reduced. Given a template, its contact graph reduction ratio is determined by how its contacts are defined. In our threading model, two residues have a contact if and only if their spatial distance is within $7\dot{A}$. If this distance limit is increased to a bigger one which often occurs in a threading model considering distance-dependent pairwise contacts, then the number of contacts within a template will also be increased. Consequently, the edge reduction ratio of template contact graphs will be bigger and the gain of graph reduction technique will be less. Conversely, if certain biological grounds can be used to prune some meaningless contacts, then the computational efficiency of our LP approach can be improved more by the graph reduction technique.

Chapter 7

RAPTOR Implementation

This chapter discusses some implementation details of our protein structure prediction server RAPTOR. In the previous chapters, we have outlined and analyzed the integer programming approach for the optimal sequence-template alignment problem. In this chapter, we first introduce the knowledge-based scoring system used in RAPTOR. The scoring system is crucial for structure prediction computer programs. Then we will describe our method for training the weight factors contained in the scoring function. Finally, we explain our Support Vector Machines (SVM) approach to fold recognition.

7.1 Scoring Systems

We calculate the averaged energy over a set of homologous sequences, as demonstrated in PROSPECT-II [62]. For the template, we use PSI-BLAST [3] to calculate the position specific mutation matrix ($PSSM$) score for each position of the template. Entry $PSSM(i, a)$ describes the mutation score of amino acid a at template position i which is the log-odds that amino acid a occurs at this position. For a query sequence of length n, an $n \times 20$ frequency matrix $PSFM$ is calculated by using PSI-BLAST with the maximum iteration number being set to five. Entry $PSFM(j, b)$ presents the occurring frequency of amino acid b at sequence position j. Assume a template position i is aligned to the sequence position j. Then the mutation score and fitness score for this position alignment are calculated as follows:

$$Mutation(i,j) = \sum_a PSFM(j,a) \times PSSM(i,a)$$

and

$$Fitness(i,j) = \sum_a PSFM(j,a) \times F(env_i, a)$$

where $F(env, a)$ describes the environment fitness potential or preference for a particular combination of amino acid type a and environment descriptor env. Here env_i denotes the environment descriptor at template position i.

We use two types of structural features to describe the local environment env of a position in the template:

- Secondary structure type. Each position of a template must be in one secondary structure. Three common classes are used: α-helix, β-strand (*beta*-sheet) and irregular structure (loop);

- Solvent accessibility (sa). Three levels are defined: buried (inaccessible), intermediate, and accessible. The boundaries between the different solvent accessibility levels are determined by the Equal-Frequency discretization method, that is, the residues in the database are equally distributed within each level. The relative solvent accessibility is used in classification. The calculated boundaries between the solvent accessibility levels are at 7% and 37%. If the solvent accessibility at this position is no more than 7%, then it is regarded as inaccessible; if the solvent accessibility is more than 37%, then it is considered as accessible; otherwise, it is intermediate.

Secondary structure and solvent accessibility assignments are generated by some software packages such as the DSSP package [57]. The combination of these two features yields nine local structural environments at each template position.

$F(env, a)$ is taken from PROSPECT-II [62] and can be calculated as follows:

$$F(env,a) = F(ss,sa,a) = -K_B T \log \frac{N(ss,sa,a)}{N_E(ss,sa,a)}$$

where K_B is the Boltzmann's constant, T is the temperature, $N(ss, sa, a)$ is the number of amino acid type a occurring in secondary structure type ss and with solvent accessibility sa

as enumerated from the database, and $N_E(ss, sa, a)$ is the expected value of $N(ss, sa, a)$, calculated as

$$N_E(ss, sa, a) = \frac{N(ss, sa)N(a)}{N}$$

where $N(ss, sa)$ is the number of residues in secondary structure type ss and with solvent accessibility sa and $N(a)$ is the number of amino acids of type a.

If the two ends of an interaction edge of the template contact graph are aligned to the j_1^{th} and j_2^{th} positions of the query sequence respectively, then the pair score for this interaction is given by

$$Pair(j_1, j_2) = \sum_a \{PSFM(j_1, a) \times \sum_b PSFM(j_2, b)P(a, b)\}$$

where $P(a, b)$ denotes the pairwise interaction potential between two amino acids a and b. P is taken from PROSPECT-I [132] and calculated by

$$P(a, b) = -K_B T \log \frac{M(a, b)}{M_E(a, b)}$$

where $M(a, b)$ is the number of contacts with one end of type a and the other end of type b, and $M_E(a, b)$ is the expected number of contacts with one end of type a and the other end of type b. The latter can be estimated by

$$M_E(a, b) = \frac{M(a)M(b)}{M}$$

where M is the total number of contacts, that is, $M = \sum_{i,j} M(i, j)$, and $M(i)$ is the number of contacts with one end being amino acid type i, that is, $M(i) = \sum_j M(i, j)$.

If template position i is aligned to sequence position j, $SS(i, j)$ is defined to be the difference between the template secondary structure at position i and the predicted sequence secondary structure at position j. We use PSIPRED [53] or other secondary structure predictors to predict the secondary structure of the query sequence. Three types of outcomes (α-helix, β-sheet and loop) of the secondary structure prediction are considered. For example, if the confidence level of α-helix, β-sheet and loop at sequence position j are $x(j)$, $y(j)$, and $z(j)$ respectively (where these sum to 1), and the secondary structure type at template position i is α-helix, then $SS(i, j) = x(j) - z(j)$. Also, if the secondary structure type at template position i is β-helix, then $SS(i, j) = y(j) - z(j)$.

As discussed in Section 4.2, in aligning the loop segment of the template to the sequence segment, the alignment gap must be penalized through our scoring function. The gap penalty function is assumed to be an affine function $b + ge$, that is, a gap open penalty b plus a length-dependent gap extension penalty ge where g is the gap length. A gap open penalty b is set at 10.6 and a gap elongation penalty e is 0.8 per single gap [41]. The alignment result is not sensitive to the values of b and e because the weight factors contained in the scoring function is trained by maximizing the overall alignment accuracy, which will be described in the next section.

7.2 Weight Training

In our scoring function (See Equation 4.1), there is a weight factor for each energy item to adjust its significance. We choose the weight factors $W_m, W_s, W_{ss}, W_g, W_p$ by optimizing the overall alignment accuracy. We select a set of 95 structurally-aligned protein pairs from Holm et al.'s test set [48] as the training samples, each of which has only a fold-level similarity, and then compare the alignments of these training pairs generated by RAPTOR with the structural alignments generated by a structure alignment program SARF [2]. An alignment for a residue is regarded as correct if it is within a shift of at most four residues away from the correct structure-structure alignment by SARF. The overall alignment accuracy is defined as the ratio between the number of correctly-aligned positions of all the threading pairs and the number of maximum alignable positions. Our objective is to maximize the overall alignment accuracy. What we need is the relative values of these weight factors; we can set one of them, say W_m, to be one. A genetic algorithm plus a local pattern search method implemented in DAKOTA [29] is used to search for the optimal values of the other weight factors. The genetic algorithm starts from forty random seeds, and terminates at forty new sets of weight factors after 300 iterations. Then the local pattern search algorithm is conducted to further improve the objective function, with the initial point being set to each of these forty new sets of weight factors. The set of weight factors with the best alignment accuracy is selected. We attain a 56% alignment accuracy over this set of training pairs. A set of 1100 protein pairs which are in the fold-level similarity is also generated from Holm et al.'s test set to test the weight factors and

50% alignment accuracy is attained. In addition, we select 95 structurally-aligned protein pairs from Holm et al.'s test set, each of which is in a superfamily-level or a family-level similarity. When the same set of weight factors is used, an 80% alignment accuracy is achieved.

During the training process, we discovered that the alignment accuracy is very robust with respect to the weight factors, that is, the alignment accuracy does not change much if the weight factors are changed. The highest alignment accuracy is attained at many sets of weight factors which are quite different. The difference of the best alignment accuracy and the worst is small, approximately 10%.

Another phenomenon is that if we classify all generated weight factors according to the similarity of the alignments of the training set, we can attain approximately $6 \sim 10$ sets of weight factors, each of which can generate about a 56% alignment accuracy for the training set; however, if we combine them together, that is, if we select the best alignment of all the alignments for each pair as the final alignment, then the alignment accuracy is improved to approximately 70%. This result shows that if we can effectively combine the prediction results of different sets of weight factors, we may be able to achieve better structure prediction performance, especially with respect to alignment accuracy.

In addition, we have also tried another objective function. Besides considering the total number of correct alignment positions, we also take into account how many alignment positions are generated by RAPTOR. Let A denote the union set of the alignment positions generated by SARF and the alignment positions generated by RAPTOR. Our objective is to maximize the standardized alignment accuracy, that is, the ratio between the total number of correct alignment positions generated by RAPTOR and the number of elements contained in A. The experimental result shows that there is no dramatic difference between these two objective functions except that the fold recognition rate can be improved by approximately 2% if the traditional z-score, which will be described in the next section, is employed to rank the templates. The possible reason is that we employ the global alignment policy to align the sequence and the template such that the set of alignment positions does not change much in the optimal situation.

7.3 Fold Recognition

Fold recognition requires an algorithm to identify the best possible templates for one target sequence. This section describes the method used in RAPTOR for fold recognition, that is, ranking all templates based on the alignments between the target sequences and the templates. It is worth mentioning that it is unnecessary to do the sequence-template alignments before conducting the fold recognition. It is also possible to classify the relationship between one sequence and one template based on certain features. So far, there are two kinds of methods used by the structure prediction community to carry out fold recognition: recognition based on the traditional z-score, and recognition by machine learning methods. Most of the current prediction programs make use of the traditional z-score to recognize the best-fit templates, whereas several programs such as Threader use a neural network to rank the templates. In RAPTOR, we use one popular machine learning method, the Support Vector Machines approach, to conduct fold recognition.

7.3.1 Traditional z-Score

After threading one pair of a sequence and a template, the z-score is calculated according to the method proposed in [13] to cancel out the composition bias. The z-score is defined to be the standard deviation of the optimal alignment score to the mean alignment score. The mean alignment score is calculated by randomly shuffling the target sequence. Let z_{raw} denote this z-score. An accurate z_{raw} can cancel out the sequence composition bias and offset the mismatch between the sequence size and the template length. However, it is time-consuming to calculate an accurate z_{raw}. Generally, it is approximately calculated by the following steps:

1. Fix the optimal alignment between the target sequence and the template;
2. Shuffle the query sequence randomly;
3. Calculate the alignment scores based on the existing alignment rather than searching the optimal alignment again;

4. Repeat the above two steps N times (N is on the order of several thousands). After the N alignment scores are obtained, we calculate their mean value S_E and standard deviation S_d. Then, z_{raw} can be calculated by the following formula:

$$z_{raw} = -\frac{S_o - S_E}{S_d}$$

where S_o is the optimal alignment score.

Compared to the SVM method, the traditional z-score is expensive to compute and has a low correct recognition rate.

7.3.2 Introduction to Support Vector Machines

Support Vector Machines and kernel methods were developed in the late 1970's by Vapnik [114]. However, it is only now that more and more researchers are paying attention to them. Here, we introduce the pattern recognition aspect of the SVM method. The idea of SVM comes from the question: Given a training task and a set of training data, what is the best learning strategy for a machine to achieve the best generalization performance? The answer is that the right balance must be struck between the accuracy attained on that particular set of training data and the "capacity" of the machine, that is, the ability of this machine to recognize any test set without errors. A machine with too much capacity remembers too many details of the training set whereas a machine with too little capacity does not capture enough features of the training data. Neither case can generalize well enough to classify the test set. The formalization of this concept leads to the SVM method. The most commonly used SVM is the nonlinear SVM. However, we will begin with the simple linear SVM because the nonlinear SVM is just a kernelized linear SVM. We briefly introduce the linear and nonlinear SVMs in the following two subsections. Burges [16] provides an excellent tutorial on the pattern recognition of SVM.

Linear Support Vector Machines

Given a set of training data $\{x_i, y_i\}$, $i = 1, 2, ..., l$, $y_i \in \{-1, 1\}$, and $x_i \in R^m$, we call x_i the data point, and y_i the class label of x_i. Assume that the training set is linear-separable; that is, the positive examples and negative examples can be separated by a hyperplane

$wx + b = 0$ (this is called a separating hyperplane). Let d^+ (d^-) denote the shortest distance from the separating hyperplane $wx + b = 0$ to the closest positive (negative) example. The "margin" of the separating hyperplane is defined to be the sum of d^+ and d^-. In order to achieve the best generalization performance, one goal is to find a hyperplane that maximizes $d = d^+ = d^-$. Without loss of generality, we can assume that the closest positive example lies in the hyperplane $wx + b = 1$, and the closest negative example lies in the hyperplane $wx + b = -1$, that is, if $y_i = 1$, then $wx_i + b >= 1$ and if $y_i = -1$, then $wx_i + b <= -1$. So, we have $d^+ = d^- = d = \frac{1}{||w||}$. The problem to find a best hyperplane becomes the following constrained optimization problem:

$$\min ||w||$$

such that

$$wx_i + b \geq 1, \quad if \quad y_i = 1$$
$$wx_i + b \leq -1, \quad if \quad y_i = -1$$

By a minor transformation, this optimization problem is equivalent to the following optimization problem:

$$\min \frac{1}{2} w^2$$

subject to

$$y_i(wx_i + b) - 1 \geq 0$$

This problem is a typical constrained convex quadratic programming (QP) problem, since the objective function is a convex function and the feasible region formed by linear constraints is also convex. The Lagrangian dual can be introduced to simplify the problem. We introduce l positive Lagrange multipliers λ_i, $i = 1, 2, ..., l$, one for each of the inequality constraints. Now the problem becomes

$$\min L_P \equiv \frac{1}{2} w^2 - \sum_{i=1}^{l} \lambda_i y_i (wx_i + b) + \sum_{i=1}^{l} \lambda_i$$

subject to

$$\lambda_i \geq 0$$

According to the Karush-Kuhn-Tucker (KKT) condition [113], a solution of the above QP is a global optimum if and only if the following conditions are satisfied:

- The derivatives of L_P with respect to w and b are zero;
- The derivatives of L_P with respect to all λ_i are zero;
- The constraints $\lambda_i \geq 0$ should not be violated by the solution.

From $\frac{\partial L_P}{\partial w} = 0$ and $\frac{\partial L_P}{\partial b} = 0$, we have

$$w = \sum_{i=1}^{l} \lambda_i y_i x_i \tag{7.1}$$

and

$$\sum_{i=1}^{l} \lambda_i y_i = 0 \tag{7.2}$$

Inserting Equation 7.1 and 7.2 into L_P yields the following dual problem of L_P:

$$\max L_D = \sum_{i=1}^{l} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j (x_i x_j)$$

subject to

$$\sum_{i=1}^{l} \lambda_i y_i = 0$$

$$\lambda_i \geq 0$$

L_P and L_D form a primal-dual pair which arrive at the optimal solution at the same value of w, b, and λ_i. L_D is a convex quadratic program; theoretically, it can be solved to any accuracy within polynomial time. However, if the training data set is large, there may be a difficulty with memory usage. Some special techniques are needed to overcome this difficulty.

In the optimal solution, the training data for which $\lambda_i > 0$ are called "support vectors". These points must lie on one of the two separating hyperplanes ($wx+b = 1$ or $wx+b = -1$) according to the strong complementary slack conditions. All the other training points with $\lambda_i = 0$ lie on neither of these two hyperplanes. Let I denote the set of i such that $\lambda_i > 0$, then we have $w = \sum_{i=1}^{l} \lambda_i y_i x_i = \sum_{i \in I} \lambda_i y_i x_i$. The whole support vector machine (SVM) is determined by these support vectors. For the new test datum $\hat{x}$, we need to calculate the sign of $\sum_{i \in I} \lambda_i y_i (\hat{x} x_i) + b$ to predict its class label. We can understand the SVM method

from the following perspective: Consider a new space Z of $|I|$ dimension, of which each support vector represents one dimension. For a test datum $\hat{x}$, in order to classify it, we first map it to a point z in the new space Z. The coordinate of z in each dimension is the projection of $\hat{x}$ on each support vector. Then $\hat{x}$ is classified by a linear classifier $\sum \lambda z + b = 0$, where λ is the vector of all nonzero λ_i.

By examining the linear separable case, we can see how to handle the linear nonseparable case. Assume that there is no hyperplane which can completely divide the training data into two parts, each of which contains data with the same class label. This assumption is more compatible with noisy"real world" data. The original model must be modified to accommodate classification errors. We introduce an error item ε_i for each training data and construct the following model:

$$\min \frac{1}{2} w^2 + C \sum_{i=1}^{l} \varepsilon_i$$

subject to

$$y_i(wx_i + b) \geq 1 - \varepsilon_i$$

$$\varepsilon_i \geq 0$$

where C is a penalty factor for classification errors and is provided by the users. A big C shows that the model allows only small errors, conversely, a small C indicates that the model can tolerate more errors. As before, we can write the following dual problem:

$$L_D \equiv \sum \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j (x_i x_j)$$

subject to

$$0 \leq \lambda_i \leq C$$

$$\sum_i \lambda_i y_i = 0$$

The only difference is that in the non-separable case, the Lagrangian multiplier is no more than the user-supplied penalty factor C. Again, w is given by $w = \sum_{i \in I} \lambda_i y_i x_i$. A test datum is classified by the same procedures.

Nonlinear Support Vector Machines

Now we generalize our linear SVM to accommodate the case where the classification function is not a linear function of the data. A very straightforward idea is to map the data points into a higher dimension space and then find a linear separator in the higher-dimension space. Assuming that the data point x_i is mapped to a point $\phi(x_i)$ in a higher dimension space, t we must solve the following dual optimization problem:

$$L_D \equiv \sum \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j (\phi(x_i)\phi(x_j))$$

subject to

$$0 \leq \lambda_i \leq C$$

$$\sum_i \lambda_i y_i = 0$$

The classifier is given by $\sum \lambda_i y_i \phi(\hat{x})\phi(x_i) + b = 0$.

Theoretically, there is no problem if we know the mapping function ϕ. However, there is a computational challenge if we calculate $\phi(x_i)$ directly, when its dimension is very large, say millions of dimensions or infinite. Notice that in the above optimization problem and classifier function, only the products $\phi(x_i)\phi(x_j)$ (and not any $\phi(x_i)$) are needed. An old trick can be used to circumvent this difficulty. If the mapping function ϕ is chosen such that the inner product of any two points in the new space can be represented as a function of the original two points, that is, there is a function K such that $\phi(x_i)\phi(x_j) = K(x_i, x_j)$, then we do not need to directly calculate $\phi(x_i)$ and $\phi(x_i)\phi(x_j)$ because we only need to compute $K(x_i, x_j)$. Function K is also called a kernel function. Thus, for the nonlinear SVM, we just need to carefully select a kernel function K. The rest is similar to the linear case. We can see why we choose to solve the dual problem rather than the original optimization problem, although both of them are convex quadratic programming problems. The key factor is that the original problem contains $\phi()$ in the constraint set such that we have to optimize the problem in the high dimension space, which is time-consuming or sometimes impossible.

There is no universally-accepted rule to select K; it is problem-specific. Generally, the kernel function must satisfy the following conditions to guarantee that the quadratic program is convex [111]:

1. $K(x, x) = 0$
2. $K(x, y) = K(y, x)$
3. $K(x, y) + K(y, z) \geq K(x, z)$

some typical kernel functions are [50, 114]:

$$K(x_1, x_2) = cx_1x_2 + c_0 \quad (7.3)$$
$$K(x_1, x_2) = (x_1x_2 + 1)^d \quad (7.4)$$
$$K(x_1, x_2) = \exp(-||x_1 - x_2||^2/2\sigma^2) \quad (7.5)$$
$$K(x_1, x_2) = tanh(ax_1x_2 + c) \quad (7.6)$$

Where Equation 7.3 is a linear kernel, Equation 7.4 is a polynomial kernel, Equation 7.5 is a radial basis function (RBF) kernel (also called Gaussian kernel), and Equation 7.6 is a multiple-layer perception kernel.

7.3.3 SVM Approach for Fold Recognition

In order to train the SVM model, we randomly chose 300 templates from the FSSP list and 200 sequences from the Holm and Sander's test set [48]. A set of 60,000 training data were formed by threading each of 200 sequences to each of 300 templates. We also used Daniel Fischer et al.'s benchmark [36] and Lindhal and Elofsson's benchmark [75] as the test sets to measure the generalization performance of our models. The relationship between one threading pair (i.e., the class label of one training data) was judged according to the SCOP database [85]. If one threading pair is in at least a fold-level similarity, then it is treated as a positive example; otherwise, it is seen as a negative example. After threading each sequence to each template, the following features are extracted from the generated optimal alignment:

- z_{raw};
- the sequence size;
- the template size;

- the number of cores in the template;
- the total core length in the template;
- the number of aligned cores;
- the number of aligned positions;
- the number of identical residues;
- the number of contacts with both ends on the aligned cores;
- the number of contacts with one end on the aligned cores and the other on the unaligned cores;
- the total alignment score;
- the mutation score;
- the singleton fitness score;
- the alignment gap penalty score;
- the secondary structure compatibility score;
- the pairwise contact potential score.

Given one threading pair (i.e., one training data or test data), our SVM model outputs a real value. A value greater than 0 indicates that this threading pair is similar to at least the fold level. However, we do not use the original output of the SVM model as the classification criterion. The original output is only used to rank the templates for one target sequence. In addition, we calculate the final z-score for each query sequence based on the SVM output. For all the threading pairs of one given sequence, let $o_1, o_2, ..., o_q$ denote the outputs from the SVM model where q is the number of templates which the sequence is threaded to. The final z-score is calculated by $\frac{o_i - u(o)}{std(o)}$, where $u(o)$ is the mean value of o_i, and $std(o)$ is the standard deviation of o_i. In our experiments, we have tried

several different kernel functions such as RBF kernel, linear kernel and polynomial kernel. The best kernel function for our data is the RBF kernel.

In the abovementioned SVM model, we have used the traditional z-score as a feature. As already mentioned, one of the advantages of the SVM method over the traditional z-score is that it is more efficient to conduct SVM classification if the SVM model is already trained. If we use the traditional z-score in the SVM model, then this advantage disappears. In fact, we have also tried the SVM model by excluding the traditional z-score. The experimental result shows that the new SVM model performs as well as the one with the traditional z-score. This indicates it is unnecessary to incorporate the traditional z-score into our SVM model.

Chapter 8

Experimental Results

In this chapter, we will summarize RAPTOR's structure prediction performance in several tests. The tests include our large scale benchmark tests, and the public and blind third-party tests by the community. There are advantages and disadvantages for each test. For the large scale benchmark tests, the advantage is that we can experiment on large scale data. Therefore, the performance of servers will not be biased due to the small amount of test data. However, we can only compare the latest performance of RAPTOR to the past performance of other servers because the standalone programs of other servers are inaccessible. As for the public and blind tests, the good point is that all the servers are evaluated at the same time by the community. The evaluation reflects all the state-of-the-art servers available to the community. But the disadvantage is that the test set is often very small due to the blindness requirement, which may cause bias. In our tests, only the fold recognition capability is measured. In the tests by the third parties, both alignment accuracy and fold recognition results are available. In terms of the sensitivity on the hard targets, RAPTOR ranks first among all the individual servers in CAFASP3, the latest worldwide protein structure prediction competition organized by the community. RAPTOR also ranks top in the latest LiveBench test, a public but not blind test.

8.1 Large Scale Benchmark Test

RAPTOR were tested on two large scale benchmarks: Fischer's benchmark and Lindahl's benchmark. The performance of all the prediction servers was measured in three different similarity levels: fold, superfamily and family. Also, their fold recognition capability was examined by evaluating the best of the first N models ($N = 1, 5, 10$). The correctness of one prediction was judged by the SCOP classification. If there is at least one correct template among the first N outputs for one target, then this target is considered to be correctly predicted in the "top N" type ($N = 1, 5, 10$).

Fischer et al.'s benchmark is relatively easy. It consists of 68 target sequences and 301 templates. RAPTOR ranks 55 positive pairs (in all the similarity levels) out of 68 pairs as the top one, achieving an 81.0% prediction rate. The detailed results are shown in Table 8.1.

Table 8.1: Fold recognition rate of RAPTOR on Fischer et al.'s benchmark. The number in this table is the number of targets correctly recognized in each type.

	All Levels	Fold Level	Superfamily Level	Family Level
top 1	55	11	24	44
top 5	58	14	29	47
top 10	59	16	29	47
# positive examples	63	23	38	48

The fold recognition performance of RAPTOR was further tested on a larger benchmark set consisting of 976 proteins [75]. By threading each one against all the others, there are 976×975 threading pairs in total. The results are shown in Table 8.2; the results of other methods are taken from Shi et al's paper [102].

As shown in Table 8.2, the performance of RAPTOR at all similarity levels is better than the others, and much better at the fold level. At the family level, RAPTOR's recognition performance is comparable to FUGUE [102], the best method for the family and superfamily levels except RAPTOR. Thus, we may conclude that a strict treatment

Table 8.2: Fold recognition rate of RAPTOR on Lindhal and Elofsson's benchmark. The number in this table is the ratio between the number of targets correctly recognized and the number of targets with a correct template in the database.

	Family		Superfamily		Fold	
Servers	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
RAPTOR	**84.8**	**87.1**	**47.0**	**60.0**	**31.3**	**54.2**
FUGUE	82.2	85.8	41.9	53.2	12.5	26.8
PSI-BLAST	71.2	72.3	27.4	27.9	4.0	4.7
HMMER-PSIBLAST	67.7	73.5	20.7	31.3	4.4	14.6
SAMT98-PSIBLAST	70.1	75.4	28.3	38.9	3.4	18.7
BLASTLINK	74.6	78.9	29.3	40.6	6.9	16.5
SSEARCH	68.6	75.7	20.7	32.5	5.6	15.6
THREADER	49.2	58.9	10.8	24.7	14.6	37.7

of the pairwise interactions is necessary for the fold level recognition or even the superfamily level. For the family level recognition, sequence (or profile) alignment can attain satisfactory results.

The benchmark results shown in Tables 8.1 and 8.2 are not as useful in evaluating RAPTOR's performance compared with other servers. This occurs because RAPTOR may have already memorized complete or partial information about each positive example because of three components implemented in RAPTOR. First, RAPTOR uses the SVM model to do fold recognition; the training data may contain some data in these benchmarks. Second, RAPTOR uses PSI-BLAST to calculate the position specific score of the templates and position specific frequency of the sequences. So, even if one pair is treated as in a fold-level similarity in the benchmarks, the sequence and the template may be linked by some intermediate sequences, which will decrease the prediction difficulty. Finally, RAPTOR uses the whole template database to calculate some parameters of the scoring function. Because there are some overlaps between the template database and the benchmarks, some structure information of the templates in the benchmarks may be memorized by RAPTOR.

Therefore, we need a better and fairer evaluation to judge RAPTOR's performance.

8.2 CAFASP3 Evaluation

In this section, we will present RAPTOR's performance in the CAFASP3 (The Third Critical Assessment of Fully Automated Structure Prediction) evaluation, the worldwide protein structure prediction competition. The goal of CAFASP is to evaluate the state-of-the-art protein structure prediction servers available to the community. Since its inception in 1998, CAFASP has been held three times, together with CASP (Critical Assessment of Structure Prediction) [84, 83] which started ten years ago. Unlike the benchmark test, the CAFASP evaluation is an absolutely blind test. Moreover, CAFASP is the largest scale blind test of structure prediction servers in the community. The experimental structures of the test sequences were unknown until the end of the test. CAFASP3 makes use of MaxSub [103] to evaluate the alignment accuracy. MaxSub first superimposes the predicted structure with the experimental structure, and then calculates the length of predicted segment which can be superimposed to the experimental structure within $5\dot{A}$ *RMSD*, and finally normalizes the length of the superimposable segment to the alignment accuracy. The recognition is considered correct only if the alignment accuracy is above a certain level. Sometimes it is easy to recognize the correct template, but it is difficult to attain the correct alignment. This evaluation policy disfavors those prediction servers with good fold recognition performance but generating bad alignments. Although each server can submit at most ten models for each target sequence, only the first model is evaluated in ranking the servers. CAFASP3 classifies all the test sequences into three different groups: Fold Recognition (FR) targets, hard Homology Modelling (HM) targets and easy Homology Modelling (HM) targets, according to the prediction difficulty of a sequence. The detailed evaluation rules are available on the CAFASP3 website (http://www.cs.bgu.ac.il/ dfischer/CAFASP3). Table 8.3 ranks all the servers in terms of the sensitivity on the FR targets (*i.e.*, the hardest targets). Table 8.4 ranks all the servers in terms of the sensitivity on the hard homology modelling targets. In these two tables, those servers whose names

are printed in italic are meta-servers[1]. As shown in these two tables, RAPTOR ranks first among all the individual servers in both alignment accuracy and fold recognition capability on FR targets. RAPTOR also ranks top in hard homology modelling target recognition. These two tables prove that RAPTOR is one of the best fold recognition servers.

Table 8.3: CAFASP3 evaluation: RAPTOR's performance on 30 Fold Recognition targets. The servers in italic are meta-servers. Sum MaxSub Score is the sum of MaxSub score of each target.

Rank	Servers	Sum MaxSub Score	# Correct
1	*3ds5 robetta*	5.17-5.25	15-17
3	*pmod 3ds3 pmode3*	4.21-4.36	13-14
4	**raptor**	**3.98**	**13**
5	shgu	3.93	13
6	*3dsn* orfeus	3.64-3.90	12-13
7	*pcons3*	3.75	12
8	fugu3 orf_c	3.39-3.67	11-12
10	fugsa orf_b	3.44-3.63	10-12
...	...	...	...
48	pdbblast	0.00	0
...	...	...	...
54	blast	0.00	0

In addition to the sensitivity of servers, the specificity is important for high-throughput automatic structure prediction programs. CAFASP3 calculates the specificity of one server according to the following procedures: (1) Rank the models submitted by one server based on the provided reliability score. Note that only the first model for each target is evaluated; (2) Count the number of correct predictions before the first K false positives $TP(K)$; (3) Calculate the average of $TP(K)$, $K = 1, 2, ..., 5$ as the specificity of the server. Table 8.5

[1]Meta-servers refer to those servers which do consensus predictions based on the outputs of other servers.

Table 8.4: CAFASP3 evaluation: RAPTOR's performance on 12 hard HM targets. The servers in italic are meta-servers. Sum MaxSub Score is the sum of MaxSub score of each target.

Rank	Servers	Sum MaxSub Score	# Correct
1	*3ds5*	5.13	12
2	*3ds3* shgu	4.93-5.02	12
4	*pmod pmod3*	4.60-4.68	12
6	orfeus orfb 3dpsm **raptor** fugu3 *pco3 robetta*	4.33-4.43	12
8	samt02	4.18	12
...	...	...	..
55	cmap	0	0

shows the specificity of all the CAFASP3 servers on 33 targets. These 33 targets consist of all the FR targets and hard HM targets. However, a multidomain FR target is treated as one target, whereas it is regarded as multiple targets in sensitivity evaluation. From this table, we can see the specificity of RAPTOR is good but not outstanding. RAPTOR's specificity is underestimated, because we tuned RAPTOR's performance until late July, 2002 and the scale of the reliability score of RAPTOR changed dramatically during the competition. In the first month of CAFASP3, the reliability score generated by RAPTOR was the original output of the SVM model, but in the following months, the reliability score was the standardized output of the SVM model.

8.3 LiveBench Evaluation

The LiveBench test is a type of test in between the CAFASPs and large scale benchmark tests. It is a public test, but not completely blind. The target sequences are submitted to the participant servers immediately after their 3D structures are released in the PDB database. Therefore, the servers updating their template databases synchronously with the PDB database can achieve better results in LiveBench. But the number of available

Table 8.5: CAFASP3 evaluation: RAPTOR's specificity on 33 targets consisting of FR targets and hard HM targets. But a multidomain FR target is counted only once. That is, if a domain of a FR target is correctly predicted, then this FR target is considered to be correctly predicted.

Servers	specificity
3ds5	24.8
pmodel 3ds3 3dsn pmode3	22.0-22.6
pcons shgu	21.4-21.6
inbgu fugu3	19.0-19.8
ffas03 fugsa orfeus	18.2-18.4
raptor 3dpsm orf_c	17.4-17.8
...	...
pdbblast	13.0
...	...
blast	4

Table 8.6: LiveBench-6 test: RAPTOR's sensitivity on easy targets.

Servers	Meta Servers	Score (Top 1)	Score (Top 10)	# Correct (Top 1)	# Correct (Top 10)
	3DS3	1156	1156	28	28
	3DS5	1156	1156	27	27
ST02		1142	1169	27	27
	RBTA	1103	1201	26	28
	PMO4	1096	1206	28	28
	PCO3	1078	1227	27	28
	PMOD	1076	1185	27	28
FFA3		1068	1167	26	27
FUG3		1060	1148	27	27
	PCO4	1053	1213	26	28
	PMO3	1048	1222	27	28
ORFs		1047	1161	27	29
	PCO2	1019	1182	27	28
FUG2		1013	1098	25	27
RAPT		1008	1084	26	27
SHGU		1007	1134	25	27
SFPP		1000	1050	24	26
3DPS		998	1126	26	28
ORFb		976	1104	25	27
INBG		947	1055	24	26
MGTH		940	1025	26	26
GETH		938	1001	24	25
PDBb		938	991	26	26
FFAS		937	995	23	24
SFAM		915	941	21	21
ST99		856	996	23	24
PCOC		851	1053	24	26
PCOB		801	1012	24	27

Table 8.7: LiveBench-6 test: RAPTOR's sensitivity on hard targets.

Servers	Meta Servers	Score (Top 1)	Score (Top 10)	# Correct (Top 1)	# Correct (Top 10)
	3DS5	992	992	32	32
	PMO3	921	1216	31	41
	PMO4	907	1186	31	41
	3DS3	890	890	29	29
	PMOD	889	1150	29	38
SHGU		830	1050	25	37
	PCO3	814	1105	27	35
	PCO2	777	1025	24	33
	RBTA	761	973	26	36
ORFs		759	995	28	36
	PCO4	693	1054	26	35
FUG2		681	903	25	30
MGTH		673	933	23	35
RAPT		665	915	22	34
3DPS		662	943	24	33
FUG3		661	865	23	30
INBG		648	942	22	36
ST02		641	755	24	29
GETH		591	769	20	25
FFA3		542	938	20	32
PCOC		483	807	19	31
PCOB		455	777	17	33
SFPP		455	703	18	27
SFAM		428	472	16	18
ORFb		424	476	16	17
ST99		369	451	14	18
FFAS		268	412	11	15
PDBb		136	192	6	9

test sequences is larger than that in the CAFASPs, and the more comprehensive evaluation criteria are used than in the CAFASPs. The LiveBench test measures the model quality from multiple perspectives, overcoming the bias of a single criterion. In this section, we summarize the latest results of the LiveBench-6 test. Here, we present only the evaluation results according to the MaxSub criterion, in order to compare them with the CAFASP3 evaluation.[2] Table 8.6 and 8.7 show the sensitivity of all servers on easy targets and hard targets respectively. The set of easy targets in the LiveBench is equivalent to the union of easy HM targets and hard HM targets in CAFASP3. Table 8.8 indicates the specificity of all the participant servers on all the targets. As shown in these three tables, RAPTOR ranks among the top non-meta servers in both sensitivity and specificity. As far as the number of correct predictions before the first false positive is concerned, RAPTOR performs as well as three individual servers ORFs, ST02, and SHGU, and most meta servers except *PMO3*, and far better than the other individual servers. But RAPTOR's specificity does not increase if more false positives are allowed. Table 8.9 shows RAPTOR's sensitivity on all targets by month. We can see RAPTOR's performance decreases from August 2002 to December 2002. The most probable reason is that the template database had not been updated since June 16, 2002.

[2]For the results evaluated by other criteria, see the website http://bioinfo.pl/LiveBench/6/.

Table 8.8: LiveBench-6 test: RAPTOR's specificity on all targets.

Servers	Meta Servers	Mean	1	2	3	4	5	6	7	8	9	10	total
	PMO3	51.3	45	48	49	50	51	54	54	54	54	54	58
	3DS5	50.0	24	46	51	51	51	53	56	56	56	56	59
	PMO4	49.6	35	42	47	50	51	53	54	54	55	55	59
	PMOD	48.1	34	39	46	48	50	52	53	53	53	53	56
	PCO3	46.3	27	44	46	47	49	49	49	50	51	51	54
ORFs		45.2	38	43	43	44	46	47	47	48	48	48	55
	3DS3	44.3	33	35	44	44	46	46	48	48	48	51	57
SHGU		43.4	35	40	41	43	44	45	46	46	47	47	50
ST02		41.8	33	38	38	40	43	43	45	46	46	46	51
	PCO2	40.9	10	32	35	42	48	48	48	48	49	49	51
FUG3		39.9	11	36	41	42	43	45	45	45	45	46	50
FUG2		39.3	13	31	34	43	43	45	45	46	46	47	50
MGTH		39.1	22	36	38	41	41	42	42	42	42	45	49
	PCO4	38.6	27	28	37	37	38	38	44	45	46	46	52
RAPT		38.6	33	34	38	40	40	40	40	40	40	41	48
FFA3		37.3	19	31	33	37	41	41	41	42	44	44	46
INBG		36.9	23	31	35	39	39	39	39	40	42	42	46
SFPP		35.3	17	23	31	33	41	41	41	42	42	42	42
GETH		33.7	28	31	32	34	34	34	34	36	37	37	44
ORFb		33.0	1	33	33	34	35	37	37	40	40	40	41
SFAM		32.5	11	24	35	35	35	37	37	37	37	37	37
ST99		32.5	26	31	32	32	34	34	34	34	34	34	37
PCOC		31.6	5	24	31	34	37	37	37	37	37	37	43
3DPS		28.6	12	14	22	26	32	35	35	36	37	37	50
PCOB		24.6	3	21	26	26	26	28	28	29	29	30	41
PDBb		23.4	1	22	22	24	25	27	27	27	29	30	32
FFAS		23.3	9	19	20	22	22	25	27	28	29	32	34
RPFD		10.1	3	5	6	7	13	13	13	13	14	14	14
BLAS		8.0	6	6	7	8	8	8	9	9	9	10	12

Table 8.9: LiveBench-6 test: RAPTOR's performance by month.

Month	MaxSub Rank
August	3
September	4
October	7
November	14
December	9

8.4 Specific Examples

In this subsection, we present some structure prediction examples generated by RAPTOR in the CAFASP3 evaluation and the LiveBench-6 test [95].

Figure 8.1 depicts the superimposition between the experimental structure (in grey) and the prediction structure (in dark grey and black) of T0136_1. The segment in dark grey is superimposable with the experimental structure within $5\dot{A}$ *RMSD*, and the rest is not superimposable. From this figure, we can see that even for such a long sequence, RAPTOR can generate a fairly good prediction.

The following two figures are generated by RasMol based on the evaluation results of LiveBench-6. Figure 8.2 shows almost a perfect structure prediction for target 1ll8A. The alignment accuracy score measured by MaxSub [103] is more than 9 (out of a possible 10). Figure 8.3 presents a good structure prediction for target 1j53A. The alignment accuracy score measured by MaxSub [103] is more than 6. Considering the length of the target sequence, we can claim that this prediction is also successful.

8.5 Discussion

RAPTOR performs well on fold family (FR) targets and hard HM targets in CAFASP3. However, it does poorly with family (HM) targets. The reason is due to three minor glitches: (1) We were still tuning the parameters late into July, 2002; (2) We had not updated our database until late June and have not constantly updated it since then; and (3) A minor error occurred while our database was being updated. We built our template database based on the FSSP list released on June 16, 2002.

For example, RAPTOR missed the HM target T0177 even though PDB-blast could predict it correctly. The best two templates for T0177 are 1konA and 1lfpA. Due to an error, 1konA had not been incorporated into our template database although it was in the FSSP list, and 1kon was released on June 19, 2002. In addition, T0177 was treated as three targets in the CAFASP3 evaluation, with the result that RAPTOR's miss on this target was amplified three-fold.

The second problem also manifested in the LiveBench-6 test after the CAFASP3 eval-

uation was that the performance of RAPTOR was decreasing. RAPTOR missed the easy target fp10972 (1khyA). Its best templates were 1k6ka and 1ksfX released in PDB on September 27, 2002.

Fixing such glitches results in a better performance not only for HM targets but also for FR targets. But since the HM targets are so easy to predict, RAPTOR's error manifested itself more glaringly here.

RAPTOR has difficulty in recognizing a positive pair due to the mismatch of the sequence length and template length. In the training pairs of the SVM model, most positive pairs contain a sequence and a template with small difference between their lengths. Thus, the resulting SVM model gives preference to the threading pair with small length difference.

When a short sequence can only be aligned to one domain of a multidomain template, RAPTOR often misses the correct prediction due to the distortion of the SVM (Support Vector Machines) model which is used to select the correct templates. An example is the LiveBench target fp9293 (1l8dA). One of the correct templates is 1i1ip. RAPTOR generates a very good alignment score between 1i1ip and fp9293 but fails to rank 1i1ip within the top ten due to the large mismatch of their lengths. The template database currently used by RAPTOR contains only chain template, not domain template. Adding domain templates to RAPTOR may be able to solve this problem because we can have a shorter template for the short sequence so that SVM can rank the template at top position.

In addition, RAPTOR has problems in predicting a long target sequence. Sometimes RAPTOR can rank several correct short templates within the top ten models, but it cannot combine these models to generate a better and longer model for the whole sequence. An example is target T0136; RAPTOR generates the best model for its first domain among all the servers, but ranks this best model as the ninth one due to length mismatch. The first model submitted by RAPTOR is also a correct prediction for the second domain of T0136. One possible solution for this problem is to split the long sequence into several domains and then build the structure for each domain respectively. The difficulty lies in how to determine the domain boundary of the sequence accurately and how to assemble multiple models to obtain a prediction for the whole sequence.

According to the CAFASP3 and LiveBench-6 results, the meta-servers are clearly leaders in both sensitivity and specificity for both easy and hard targets. This is not surprising,

but the big question remains: what is the most important factor for the success of meta servers?

Figure 8.1: Superimposition of the experimental structure (in grey) and the predicted structure (in dark grey and black) of CAFASP3 target T0136_1. The segment in dark grey is superimposable with the experimental structure within $5\dot{A}$ *RMSD*, and the segment in black is not superimposable. (The figure is taken from the CAFASP3 website.)

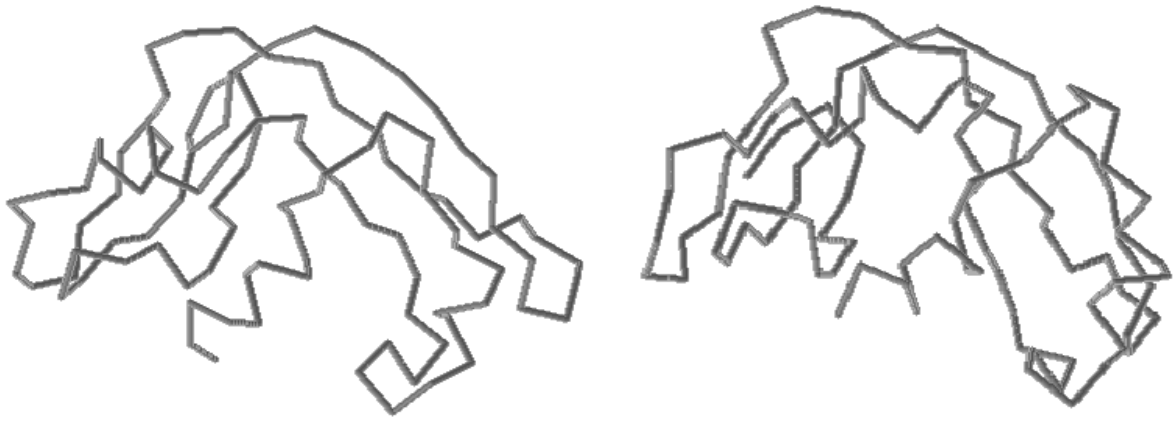

Figure 8.2: Experimental structure (left) and predicted structgure (right) of 1kvzA.

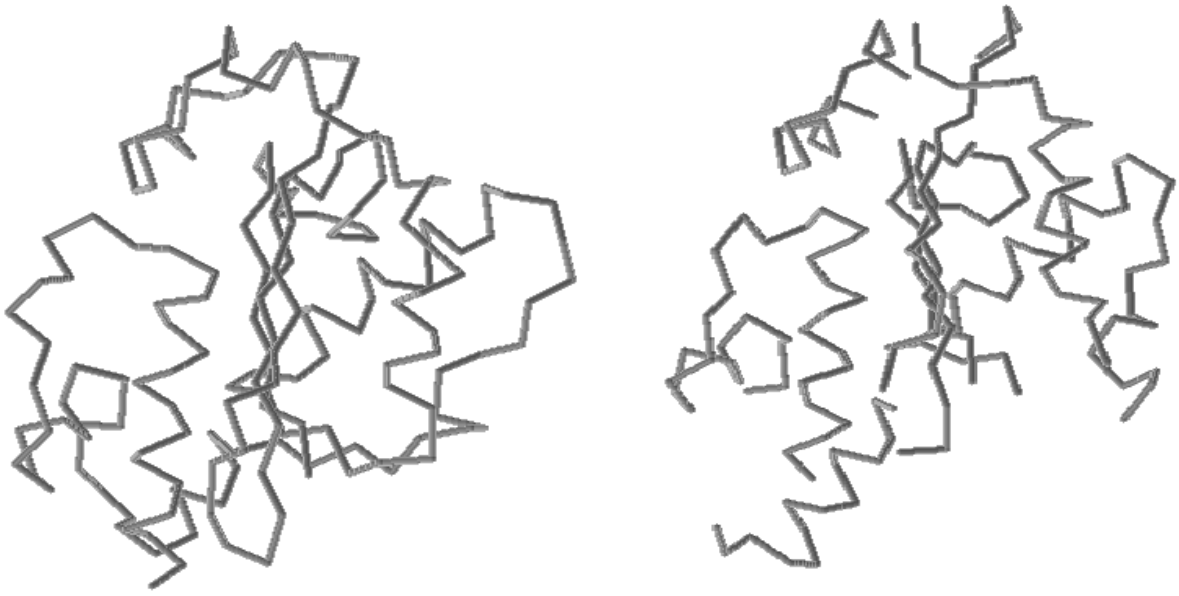

Figure 8.3: Experimental structure (left) and predicted structure (right) of 1j53A.

Chapter 9

Conclusions and Future Work

9.1 Conclusions

This thesis demonstrates that the linear programming approach is a powerful tool for solving the protein threading problem. It has previously been suggested that this problem is intractable. Intractability is caused by two key factors of the threading model: the pairwise contact potential contained in the scoring function and the variable gaps allowed within the alignments.

First, based on the contact map graph of the protein structural templates, we formulated the protein threading problem as three integer programs with constraint matrix sizes ranging from dozens of rows and columns to millions. Also, we compared the strengths of these three formulations in terms of the size of their solution spaces. The simplicity of the formulations enables researchers with some knowledge of linear and integer programming to easily implement and test them with their own parameters, provided that they can access a linear and integer programming software package. The experimental results demonstrate that almost all protein threading instances can be solved by using only linear programming solvers (no branch-and-bound method is needed), indicating that the protein threading problem is tractable in practice.

We have conducted some theoretical analysis of the linear solutions of our formulation based on polyhedral combinatorics theory, to gain some insight concerning why almost all solutions found in practice are integral. The cutting plane analysis suggests that our linear

program formulation to the protein threading problem is very effective due to the special structure of the problem, that is, our formulation successfully exploits the special structure of the threading problem.

The number of variables and non-zero elements in the constraint matrix is proportional to the number of edges in the template contact graph. For a threading pair of a template with a complex contact graph and a long sequence, the number of variables and nonzero elements in the constraint matrix of our formulation is often large, sometimes several millions, which will result in the linear program solvers to converge to the optimal solution in a long time. In light of these cases, we develop a graph reduction technique to minimize the number of variables and non-zero elements by reducing a template contact graph to a new one with fewer edges. Our graph reduction technique comes from the following two observations: (1) If the spatial neighbourhood distance cutoff is fixed to $7\dot{A}$, many templates contain some subsegments which have a regular contact subgraph; (2) For those templates with a regular contact graph, the divide-and-conquer algorithm developed by Xu et al. [132, 129] can outperform the linear programming approach. Therefore, we use the divide-and-conquer algorithm to align the subsegments with regular contact subgraphs to the target sequences first. A regular subgraph is reduced to a single edge, and the original contact graph is reduced to a new one with fewer edges. Finally, the linear programming approach is used to align the template to the sequence based on the reduced contact graph. The experimental results show that the more the template contact graph is reduced, the greater the computational efficiency can be improved and that the computational efficiency of threading a long sequence to the whole template database can be improved by an average at 30%. One fact worthwhile to mention is that this kind of speedup is achieved without losing any prediction accuracy because the same scoring function is still globally optimized. We are also developing another method to speed up RAPTOR as follows. First, we adopt the similar technique described in PROSPECT-II [62] to rank all the templates and select top N (several hundred) templates to out of the template database. Second, we employ the linear programming approach to align the sequence to the selected templates one by one and select the best templates with the best overall alignment quality. This method can greatly improve the structure prediction efficiency. But we need to carefully choose the value of N to avoid large accuracy loss.

In contrast to the traditional time-consuming z-score method to recognize the best-fit templates, we use the mathematically-grounded Support Vector Machines approach to select the best templates for each target sequence. The experimental results indicate that the SVM model can improve the template selection rate by approximately 5%. We expect that an improved SVM model will do better in the template selection.

9.2 Future Work

Our novel integer programming approach to protein threading opens up many avenues for future research in protein structure prediction.

Our linear program formulation can be easily extended to incorporate other constraints such as those derived from sparse NMR data, preliminary experiments, and biologists' insight. For a large protein, the experimental NMR method often fails to determining its structure, but can yield some sparse NMR data [130]. Based on these sparse NMR data, we can infer some distance constraints among some of the protein residues. Generally, two kinds of distance constraints can be formed: hard constraints and soft constraints. A hard distance constraint place an upper bound on the spatial distance of the two sequence residues. A soft constraint gives a reward for bringing two residues closer. For each hard constraint, we can formulate it as a linear constraint and then add the constraint to our linear program formulation. For a soft constraint, we can add one variable for it and a certain potential score can be developed to favor it. Therefore, it is very natural to extend our linear program formulations to incorporate both hard and soft constraints.

Due to parameter estimation errors in the scoring function, the minimal scoring function value does not necessarily correspond to the best alignment between the sequence and the template. One strategy is that we can generate the top N, (say $N = 100$), alignments corresponding to the lowest N objective scoring function values, and then employ some structure verification computer programs such as PROCHECK [81, 64] and WHATIF [116] to select the best alignment. Of course, we can search the N alignments N times by invalidating the previously generated alignments at each iteration. But it is very expensive in terms of computational time, if we repeatedly do N threadings from scratch. The Simplex method of linear programming is suited for this task. After generating an valid

alignment, we can add one linear constraint into the formulation to invalidate it. We do not need to solve the new linear program from scratch; if a linear program is already solved by the Simplex method, then its dual is still feasible after we add one extra linear constraint into it. This indicates that we can use the dual simplex method to solve the new linear program starting from the solution of the original linear program. [21, 98]. Therefore, the Simplex method allows us easily to generate the N solutions corresponding to the lowest N scoring function values inexpensively.

The linear programming approach is also suitable for interactive protein structure prediction. If users are not satisfied with the initial alignment between a sequence and a template, they can add some extra constraints to the original formulation based on their insight or observation. For example, they can specify the template positions to which some sequence residues should be aligned. they can also specify whether or not two sequence residues should be spatially nearby. These kind of observations can be easily formulated as linear constraints. As already mentioned, we do not need to rerun the linear program solver from scratch if some extra constraints are introduced. The dual simplex method can effectively reuse the last solution, and find the new alignment very quickly.

Many sequence-structure motifs have been discovered by motif finding tools such as Trilogy [10]. The sequence-structure motifs describe the relationship among multiple residues. This kind of relationship can also be easily formulated and included in our hypergraph-based linear program.

We could apply more sequence-structure conservation features to improve the prediction rate and scale down the linear program size. Cores are the most conserved segments of a template sequence. They occur among all the homologs of the template protein. The residue composition of a core might not be random. By carefully analyzing the residue composition of a core, we may be able to exclude some possible alignment positions of this core, and as such, decrease the number of variables needed. For example, Baker's I-site library stores all the known structure fragments for a sequence segment of $3 \sim 9$ residues [17]. Incorporating these information into our program may not only improve our computational efficiency, but also the prediction accuracy as well.

A good structure prediction program consists of an efficient algorithm and a good scoring function. For different targets, we need different scoring functions to measure

the alignment quality. For easy HM targets, a scoring function containing only sequence information (mutation potential) is enough, but this kind of scoring function cannot recognize FR targets. For hard HM targets, the mutation information will be relaxed to the sequence-profile information such as the PSSM (Position Specific Score Matrix) generated by PSI-BLAST [3], or the multiple alignments. The environment fitness potential will be introduced into the scoring function. For FR targets, we need to further relax the mutation potential, and add more structure information such as the pairwise contact potential into the scoring function. In our scoring function, we have already incorporated the pairwise potentials; however, we still use the sequence-profile information generated by PSI-BLAST. The CAFASP3 evaluation [37] shows that this scoring function was not sensitive enough to the hard FR targets. The next step is to relax the sequence-profile information to the structure-profile information, and enhance the pairwise contact potential in the scoring function, in attempt to improve the sensitivity on hard FR targets.

We can also explore the multiple sets of weight factors for the scoring function. In the current implementation of RAPTOR, we use only a set of weight factors for all the threading pairs. It is impossible to capture well the relationship among these energy items with only one set of weight factors for the various protein pairs. In addition, experiments show that if we use six sets of weight factors to align each threading pairs, and then select the best alignment as the final one of this pair, then the overall alignment accuracy can be improved by approximately 10%. Based on this observation, a straightforward idea is to improve structure prediction performance (both the fold recognition and alignment accuracy) by using multiple sets of weight factors. We need to solve two problems for this strategy: (1) how to select the best alignment from the multiple set of alignments resulting from multiple scoring functions; and (2) how to avoid repeatedly searching for the optimal alignment for each scoring function corresponding to each set of weight factors, based on our linear program formulation.

The model assembly technique for multidomain proteins is important. For a long target sequence of multidomains, more often than not, there is no a single template in the database which can match the whole sequence. RAPTOR often ranks several single-domain templates within the top ten, each of which can match a segment of the target sequence. Based on any single template, we can only construct the coordinates for the

residues in a segment of the sequence. How to assemble these fragment models built from the single-domain templates into a complete model continues to be an interesting and challenging problem.

Finally, we can incorporate multiple-body contact potential. With the increase of available experimental protein structures, it is possible to accurately estimate the contact potential among three residues or more. Although the computational efficiency will deteriorate, the hypergraph-based linear program formulations discussed in Chapter 6 can be used to formulate this case.

Bibliography

[1] T. Akutsu and S. Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 210:261–275, 1999.

[2] N.N. Alexandrov. SARFing the PDB. *Protein Engineering*, 9:727–732, 1996.

[3] S.F. Altschul, T.L. Madden, A.A. Schaffer, J. Madden, A.A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D.J. Lipman. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research*, 25:3389–3402, 1997.

[4] C.B. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–238, 1973.

[5] NIAS DNA Bank. Growth of weekly updates of protein sequence databases, 2003. http://www.dna.affrc.go.jp/htdocs/growth/P-daily.html.

[6] Protein Data Bank. http://www.rcsb.org/pdb/holdings.html, 2003.

[7] J.E. Beasley. *Advances in Linear and Integer Programming*. Oxford University Press, Oxford, UK, 1996.

[8] B. Berger and G. Ramachandran. Introduction to computational molecular biology, 2001. course notes Lecture 17.

[9] J.U. Bowie, R. Luthy, and D. Eisenberg. A method to identify protein sequences that fold into a known three dimensional structure. *Science*, 253:164–170, 1991.

[10] P. Bradley, P.S. Kim, and B. Berger. Trilogy: discovery of sequence-structure patterns across diverse proteins. In *Proceedings of the Seventh Annual International Conference on Research in Computational Molecular Biology*, pages 77–88, 2002.

[11] C. Branden and J. Tooze. *Introduction to Protein Structure*. Garland Publishing Inc., NY, USA, 1991.

[12] C.L. Brooks, M. Karplus, and B.M. Pettitt. *Proteins: A theoretical perspective of dynamics, structure and thermodynamics*. John Wiley and Sons, New York, 1990.

[13] S.H. Bryant and S.F. Altschul. Statistics of sequence-structure threading. *Current Opinion in Structural Biology*, 5:236–244, 1995.

[14] S.H. Bryant and C.E. Lawrence. An empirical energy function for threading protein sequence through folding motif. *Proteins: Structure, Function and Genetics*, 16:92–112, 1993.

[15] J.M. Bujnicki, A. Elofsson, D. Fischer, and L. Rychlewski. Livebench-2: large-scale automated evaluation of protein structure prediction servers. *Proteins: Structure, Functions and Genetics*, S5:184–191, 2001.

[16] C.J.C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.

[17] C. Bystoff, K. Simons, K. Han, and D. Baker. Local sequence-structure correlations in proteins. *Current Opinion in Biotechnology*, 7:417–421, 1996.

[18] T. Cardozo, M. Totrov, and R. Abagyan. Homology modelling by the ICM method. *Proteins: Structure, Function and Genetics*, 23:403–414, 1995.

[19] R. Carr, G. Lancia, and S. Istrail. On the integer programming approach to independent set problems with applications to protein structure alignment. Technical Report SAND2000-2171, Sandia National Labs, Albuquerque, 2000.

[20] C. Chothia and A.M. Lesk. The relation between the divergence of sequence and structure in proteins. *EMBO Journal*, 5:823–836, 1986.

[21] V. Chvátal. *Linear Programming*. W.H. Freeman, NY, USA, 1983.

[22] International Human Genome Sequencing Consortium. Initial sequencing and analysis of the human genome. *Nature*, 409:860–921, 2001.

[23] F. Corpet, F. Servant, J. Gouzy, and D. Kahn. ProDom and ProDom-CG: tools for protein domain analysis and whole genome comparisons. *Nucleic Acids Research*, 28:267–269, 2000.

[24] G.B. Dantzig. *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ, 1963.

[25] M.O. Dayhoff, W.C. Barker, and L.T. Hunt. Establishing homologies in protein sequences. *Meth. Enzym.*, 91:524–545, 1983.

[26] M.O. Dayhoff and R.V. Eck. A model of evolutionary changes in proteins. In M.O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 3, pages 33–41. National Biomedical Research Foundation, Washington D.C., 1968.

[27] R. Dorfman, P.A. Samuelson, and R.M. Solow. *Linear Programming and Economic Analysis*. McGraw-Hill, NY, USA, 1987.

[28] Y. Duan and P. Kollman. Pathways to a protein folding intermediate observed in a 1-microsecond simulation in aqueous solution. *Science*, 282:740–744, 1998.

[29] M.S. Eldred, A.A. Giunta, B.G. van Bloemen Waanders, S.F. Wojtkiewicz, W.E. Hart, and M.P. Alleva. DAKOTA, a multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis. Version 3.0 user manual. Technical Report SAND2001-3796, Sandia, 2002.

[30] H. Fang, B. Wallner, J. Lundstrom, C. von Wowern, and A. Elofsson. Improved fold recognition by using the Pcon consensus approaches. In *Protein Structure Prediction: Bioinformatics Approach*, July 2002.

[31] U. Feige, G. Kortsarz, and D. Peleg. The dense k-subgraph problem. *Algorithmica*, 29(3):410–421, 2001.

[32] S. Fields. Proteomics: Proteomics in genomeland. *Science*, 291:1221–1224, 2001.

[33] A.L. Fink. Chaperone-mediated protein folding. *Physiological Reviews*, 79:425–449, 1999.

[34] D. Fischer. Hybrid fold recognition: Combining sequence derived properties with evolutionary information. In R.B. Altman, A.K. Dunker, L. Hunter, T.A. Jung, and T.E. Klein, editors, *Proceedings of the 2000 Pacific Symposium Biocomputing*, pages 119–130, Hawaii, January 2000. World Scientific Inc.

[35] D. Fischer. 3D-SHOTGUN: A novel, cooperative, fold-recognition meta predictor. *Proteins: Structure, Function and Genetics*, 51:434–441, 2003.

[36] D. Fischer, A. Elofsson, J.U. Bowie, and D. Eisenberg. Assessing the performance of fold recognition methods by means of a comprehensive benchmark. In L. Hunter and T. Klein, editors, *Proceedings of the 1996 Pacific Symposium on Biocomputing*, pages 300–318, Singapore, 1996. World Scientific Publishing Co.

[37] D. Fischer, L. Rychlewski, R.L. Dunbrack, A.R. Ortiz, and A. Elofsson. CAFASP3: The third critical assessment of fully automated structure prediction methods, 2003.

[38] K. Ginalski, A. Elofsson, D. Fischer, and L. Rychlewski. 3D-Jury: A simple approach to improve protein structure predictions. *Bioinformatics*, 19:1015–1018, 2003.

[39] A. Godzik, A. Kolinski, and J. Skolnick. A topology fingerprint approach to inverse protein folding problem. *Journal of Molecular Biology*, 227:227–238, 1992.

[40] A. Godzik and J. Skolnick. Sequence-structure matching in globular proteins: application to supersecondary and tertiary structure determination. *Proceeding of National Academy of Sciences*, 89:12098–12102, 1992.

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

[42] N. Guex and M.C Peitsch. Swiss-model and the swiss-pdbviewer: An environment for comparative protein modelling. *Electrophoresis*, 18:2714–2723, 1997.

[43] J. Guo, D. Kim, D. Xu, and Y. Xu. Improving the performance of DomainParser for structural domain partition using neural network. *Nucleic Acids Research*, 31(3):1–9, 2002.

[44] R.W. Harrison, D. Chatterjee, and I.T. Weber. Analysis of six protein structures predicted by comparative modelling techniques. *Proteins: Structure, Function and Genetics*, 23:463–471, 1995.

[45] T.F. Havel and M.E. Snow. A new method for building protein conformations from sequence alignments with homologues of known structure. *Journal of Molecular Biology*, 217:1–7, 1991.

[46] S. Henikoff and J.G. Henikoff. Amino acid substitution matrices from protein blocks. *Proceedings of National Academy of Sciences*, 89:10915–10919, 1992.

[47] L. Holm and C. Sander. Mapping the protein universe. *Science*, 273:595–602, 1996.

[48] L. Holm and C. Sander. Decision support system for the evolutionary classification of protein structures. In *Proceedings of the Fifth International Conference on Intelligent Systems for Molecular Biology*, pages 140–146, 1997.

[49] T. Jiang and L. Wang. Algorithmic methods for multiple sequence alignment. In *Current Topics in Computational Molecular Biology*. MIT Press, 2002.

[50] T. Joachims. *Learning to Classify Text Using Support Vector Machine*. PhD thesis, University of Dortmund, 2002.

[51] M.S. Johnson and J.P. Overington. A structural basis for sequence comparisons-an evaluation of scoring methodologies. *Journal of Molecular Biology*, 233:716–738, 1993.

[52] D.T. Jones. GenTHREADER: An efficient and reliable protein fold recognition method for genomic sequences. *Journal of Molecular Biology*, 287:797–815, 1999.

[53] D.T. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *Journal of Molecular Biology*, 292:195–202, 1999.

[54] D.T. Jones. Predicting novel protein folds by using FRAGFOLD. *Proteins: Structures, Functions and Genetics*, Supplement 5:127–132, 2002.

[55] D.T. Jones, W.R. Taylor, and J.M. Thornton. A new approach to protein fold recognition. *Nature*, 358:86–98, 1992.

[56] T.A. Jones and S. Thirup. Using known substructures in protein model building and crystallography. *EMBO Journal*, 5:819–823, 1986.

[57] W. Kabsch and C. Sander. Dictionary of protein secondary structure: protein recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22:2577–2637, 1983.

[58] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[59] K. Karplus, C. Barrett, M. Cline, M. Diekhans, L. Grate, and R. Hughey. Predicting protein structure using only sequence information. *Proteins: Structure, Function and Genetics*, S3:121–125, 1999.

[60] L.A. Kelley, R.M. MacCallum, and M.J.E. Sternberg. Enhanced genome annotation using structural profiles in the program 3D-PSSM. *Journal of Molecular Biology*, 299(2):499–520, 2000.

[61] L.G. Khachian. A polynomial algorithm in linear programming. *Dokl. Akad. Nauk SSSR*, 244:1093–1096, 1979. English translation in Soviet Math. Dokl. 20,191-194,1979.

[62] D. Kim, D. Xu, J. Guo, K. Ellrott, and Y. Xu. PROSPECT II: Protein structure prediction method for genome-scale applications. 2002. manuscript.

[63] M.A. Kurowski and J.M. Bujnicki. GeneSilico protein structure prediction meta-server. *Nucleic Acids Research*, 31(13):3305–3307, 2003.

[64] R.A. Laskowski, M.W. MacArthur, D.S. Moss, and J.M. Thornton. PROCHECK: a program to check the stereochemical quality of protein structures. *Journal of Applied Crystallography*, 26:283–291, 1993.

[65] R. Lathrop, R. Rogers, T. Smith, and J. White. A bayes-optimal probability theory that unifies protein sequence-structure recognition and alignment. *Bulletin of Mathematical Biology*, 60:1039–1071, 1998.

[66] R. H. Lathrop, R. G. Rogers Jr., J. Bienkowska, B.K.M. Bryant, L.J. Buturovic, C. Gaitatzes, R. Nambudripad, J. V. White, and T.F. Smith. *Computational Methods in Molecular Biology*, chapter 12, pages 227–283. Elsevier Press, Amsterdam, 1998.

[67] R.H. Lathrop. The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Engineering*, 7:1059–1068, 1994.

[68] R.H. Lathrop and T.F. Smith. A branch-and-bound algorithm for optimal protein threading with pairwise (contact potential) amino acid interactions. In *Proceedings of the 27th Hawaii International Conference on System Sciences*. IEEE Computer Society Press, 1994.

[69] R.H. Lathrop and T.F. Smith. Global optimum protein threading with gapped alignment and empirical pair score functions. *Journal of Molecular Biology*, 255:641–665, 1996.

[70] C.A. Laughton. Prediction of protein side-chain conformations from local three dimensional homology relationships. *Journal of Molecular Biology*, 235:1088–1097, 1994.

[71] R.H. Lee. Protein model building using structural homology. *Nature*, 356:543–544, 1992.

[72] A.M. Lesk and D.R. Boswell. Homology modelling: inferences from tables of aligned sequences. *Current Opinion in Structural Biology*, 2:242–247, 1992.

[73] M.A. Lesk. *Introduction to Protein Architecture*. Oxford University Press, Oxford, UK, 2001.

[74] W. Li, F. Pio, K. Pawlowski, and A. Godzik. Saturated BLAST: detecting distant homology using automated multiple intermediate sequence BLAST search. *Bioinformatics*, 16:1105–1110, 2000.

[75] E. Lindahl and A. Elofsson. Identification of related proteins on family, superfamily and fold level. *Journal of Molecular Biology*, 295:613–625, 2000.

[76] J. Lundstrom, L. Rychlewski, J. Bujnicki, and A. Elofsson. A neural-network based consensus predictor that improves fold recognition. *Protein Science*, 10(11):2354–2362, 2001.

[77] A. Z. Machalek. From genes to proteins: Nigms catalogs the shapes of life. *NIH Record*, 53(4), Feburary 2001.

[78] T. Madej, J.F. Gibrat, and S.H. Bryant. Threading a database of protein cores. *Proteins: Structure, Function and Genetics*, 23:356, 1995.

[79] J. McCammon and S. Harvey. *Dynamics of Proteins and Nucleic Acids*. Cambridge University Press, Cambridge, UK, 1987.

[80] L.A. Mirny, A. Finkelstein, and E. Shakhnovich. Statistical significance of protein structure prediction by threading. *Proceedings of the National Academy of Sciences*, August 2000.

[81] A.L. Morris, M.W. MacArthur, E.G. Hutchinson, and J.M. Thornton. Stereochemical quality of protein structure coordinates. *Proteins: Structure, Function and Genetics*, 12:345–364, 1992.

[82] S. Mosimann, R. Meleshko, and N.G. James. A critical assessment of comparative molecular modelling of tertiary structures of proteins. *Proteins: Structure, Function and Genetics*, 23:301–317, 1995.

[83] J. Moult, K. Fidelis, A. Zemla, and T. Hubbard. Casp5, 2002. http://predictioncenter.llnl.gov/casp5/Casp5.html.

[84] J. Moult, J.T. Pedersen, R. Hudson, and K. Fidelis. A large scale experiment to assess protein structure prediction methods. *Proteins: Structure, Function and Genetics*, 23:2–4, 1995.

[85] A.G. Muzrin, S.E. Brenner, T. Hubbard, and C. Chothia. SCOP: a structural classification of proteins database for the investigation of sequences and structures. *Journal of Molecular Biology*, 247:536–540, 1995.

[86] S.B. Needleman and C.D. Wunsch. A general method applicable to the search for similarities in the amino acid sequences of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[87] C. Orengo. Classification of protein folds. *Current Opinion in Structural Biology*, 4:429–440, 1994.

[88] C.A. Orengo, A.D. Michie, S. Jones, D.T. Jones, M.B. Swindells, and J.M. Thornton. CATH-A hierarchic classification of protein domain structures. *Structure*, 5:1093–1108, 1997.

[89] J.P. Overington. Comparison of three-dimensional structures of homologous proteins. *Current Opinion in Structural Biology*, 2:394–401, 1992.

[90] F.M.G. Pearl, D. Lee, J.E. Bray, I. Sillitoe, A.E. Todd, A.P. Harrison, J.M. Thornton, and C.A. Orengo. Assigning genomic sequences to CATH. *Nucleic Acids Research*, 28:277–282, 2000.

[91] D.R. Ripoll and H.A. Scheraga. On the multiple minima problem in the conformational analysis of polypeptides. *Biopolymers*, 30:165–176, 1990.

[92] B. Rost. Twilight zone of protein sequence alignments. *Protein Engineering*, 12:85–94, 1999.

[93] B. Rupp. Protein structure basics, 2000. http://www-structure.llnl.gov/Xray/tutorial/protein_structure.htm.

[94] R.B. Russell and G.J. Barton. Multiple protein structure alignment from tertiary structure comparison: assignment of global and residue confidence levels. *Proteins: Structure, Function and Genetics*, 14:309–323, 1992.

[95] L. Rychlewski, D. Fischer, and A. Elofsson. LiveBench-6: Large scale evaluation of protein structure prediction servers, 2003. in press.

[96] A. Sali and T.L. Blundell. Comparative protein modelling by satisfaction of spatial restraints. *Journal of Molecular Biology*, 234:779–815, 1993.

[97] H. Schrauber, F. Eisenhaber, and O. Argos. Rotamers, to be or not to be? *Journal of Molecular Biology*, 230:592–612, 1993.

[98] A. Schrijver. *Theory of Linear and Integer Programming.* John Wiley and Sons, 1986.

[99] R.M. Schwartz, M.O. Dayhoff, and B.C. Orcutt. Matrices for measuring distant relationship. In M.O. Dayhoff, editor, *Atlas of Protein Sequence and Structure*, volume 5, pages 353–358. National Biomedical Research Foundation, Wahington, D.C., 1978.

[100] F. Servant, C. Bru, S. Carrere, E. Courcelle, J. Gouzy, D. Peyruc, and D. Kahn. ProDom: Automated clustering of homologous domains. *Bioinformatics*, 3(3):246–251, 2002.

[101] L. Shepp. Linear programming in tomography, probability and finance. Technical Report DIMACS TR: 97-67, USA, 1997.

[102] J. Shi, L. B. Tom, and M. Kenji. FUGUE: Sequence-structure homology recognition using environment-specific substitution tables and structure-dependent gap penalties. *Journal of Molecular Biology*, 310:243–257, 2001.

[103] N. Siew, A. Elofsson, L. Rychlewski, and D. Fischer. MaxSub: An automated measure for the assessment of protein structure prediction quality. *Bioinformatics*, 16(9):776–785, 2000.

[104] K. Simons, R. Bonneau, I. Ruczinski, and D. Baker. Ab initio protein structure prediction of casp iii targets using rosetta. *Proteins: Structure, Function and Genetics*, S3:171–176, 1999.

[105] J. Skolnick, A. Kolinski, D. Kihara, M.R. Betancourt, P. Rotkiewicz, and M. Boniecki. Ab initio protein structure prediction via a combination of threading, lattice folding, clustering and structure refinement. *Proteins: Structure, Function and Genetics*, 5:149–156, 2001.

[106] T.F. Smith and M.S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[107] N.L. Summers and M. Karplus. Modelling of side chains, loops and insertions in proteins. *Meth. Enzym.*, 202:156–205, 1991.

[108] M.B. Swindells and J.M. Thornton. Modelling by homology. *Current Opinion in Structural Biology*, 1:219–223, 1991.

[109] R. Thiele, R. Zimmer, and T. Lenguaer. Protein threading by recursive dynamic programming. *Journal of Molecular Biology*, 290:757–779, 1999.

[110] W. van Gunsteren. Validation of molecular dynamics simulation. *Journal of Computational Physics*, 108:6109–6116, 1998.

[111] R.J. Vanderbei. Symmetric quasi-definite matrices. *SIAM Journal of Optimization*, 5(1):100–113, 1995.

[112] R.J. Vanderbei. *Linear Programming: Foundations and Extensions.* Kluwer Academic Publisher, Boston, USA, 2001.

[113] R.J. Vanderbei. *Linear Programming: Foundations and Extensions*, chapter 18, pages 307–313. Kluwer Academic Publisher, Boston, USA, 2001.

[114] V.N. Vapnik. *The Nature of Statistical Learning Theory.* Springer, Berlin, 1995.

[115] T. Vossen, M. Ball, and R.H. Smith. On the use of integer programming models in ai planning. In *International Joint Conference on Artificial Intelligence*, pages 304–309, 1999.

[116] G. Vriend. WHATIF: a molecular modeling and drug design program. *Journal of Molecular Graphics*, 8:52–56, 1990.

[117] J.E. Wampler. Tutorial on peptide and protein structure, 1996. http://bmbiris.bmb.uga.edu/wampler/tutorial/prot1.html.

[118] S.J. Weiner, P.A. Kollman, D.A.Case, U.C.Singh, C.Ghio, G.Alagona, S.Profeta, and P.Weiner. A new force field for molecular mechanical simulation of nucleic acids and proteins. *Journal of the American Chemistry Society*, 106:765–784, 1984.

[119] S.J. Wheelan, A.M. Bauer, and S.H. Bryant. Domain size distributions can predict domain boundaries. *Bioinformatics*, 16(7):613–618, 2000.

[120] G. Wider. Structure determination of biological macromolecules in solution using NMR spectroscopy. *BioTechniques*, 29:1278–1294, 2000.

[121] M. Wilmanns and D. Eisenberg. Inverse protein folding by the residue pair preference profile method. *Protein Engineering*, 8:626–639, 1995.

[122] D.G. Wilson and B.D. Rudin. Introduction to the optimization subroutine library. *IBM Systems Journal*, 31(1):4–10, 1992.

[123] L.A. Wolsey. *Integer Programming*. John Wiley and Sons, 1998.

[124] D. Xu, M.A. Unseren, Y. Xu, and E.C. Uberbacher. Sequence-structure specificity of a knowledge based energy function at the secondary structure level. *Bioinformatics*, 16(3):257–268, 2000.

[125] J. Xu. Speedup LP approach to protein threading by graph reduction. In *Lecture Notes in Computer Science*, Budapest, Hungary, September 2003. The Third Workshop of Algorithms in Bioinformatics.

[126] J. Xu and M. Li. Assessment of raptor's linear programming approach in CAFASP3. *Proteins: Structure, Function and Genetics*, 2003. in press.

[127] J. Xu, M. Li, D. Kim, and Y. Xu. RAPTOR: Optimal protein threading by linear programming. *Journal of Bioinformatics and Computational Biology*, 1(1):95–117, 2003.

[128] J. Xu, M. Li, G. Lin, D. Kim, and Y. Xu. Protein threading by linear programming. In *Proceedings of 2003 Pacific Symposium on Biocomputing*, pages 264–275, Hawaii, USA, 2003.

[129] Y. Xu and D. Xu. Protein threading using PROSPECT: design and evaluation. *Protein: Structure, Function and Genetics*, 40:343–354, 2000.

[130] Y. Xu, D. Xu, O. Crawford, and J.R. Einstein. A computational method for NMR-constrained protein threading. *Journal of Computational Biology*, 7:449–467, 2000.

[131] Y. Xu, D. Xu, and H.N. Gabow. Protein domain decomposition using a graph-theoretic approach. *Bioinformatics*, 16(12):1091–1104, 2000.

[132] Y. Xu, D. Xu, and E.C. Uberbacher. An efficient computational method for globally optimal threadings. *Journal of Computational Biology*, 5(3):597–614, 1998.

[133] N. Yanev and R. Andonov. Protein threading is in P? Technical Report INRIA RR-4577, French, October 2002.