2005

Parallel evolution strategy for protein threading.

Md. Rafiqul Islam
University of Windsor

Parallel Evolution Strategy for Protein Threading

By

Md. Rafiqul Islam

A Thesis
submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science
in Partial Fulfilment of the Requirements for
the Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada
2005

Your file Votre référence
ISBN: 0-494-09831-7
Our file Notre référence
ISBN: 0-494-09831-7

Abstract

A protein-sequence folds into a specific shape in order to function in its aqueous state. If the primary sequence of a protein is given, what is its three dimensional structure? This is a long-standing problem in the field of molecular biology and it has large implication to drug design and cure. Among several proposed approaches, protein threading represents one of the most promising technique. The protein threading problem (PTP) is the problem of determining the three-dimensional structure of a given but arbitrary protein sequence from a set of known structures of other proteins. This problem is known to be NP-hard and current computational approaches to threading are time-consuming and data-intensive. In this thesis, we proposed an evolution strategy (ES) based approach for protein threading (EST).

We also developed two parallel approaches for the PTP problem and both are parallelizations of our novel EST. The first method, we call SQST-PEST (Single Query Single Template Parallel EST) threads a single query against a single template. We use ES to find the best alignment between the query and the template, and ES is parallelized. The second method, we call SQMT-PEST (Single Query Multiple Templates Parallel EST) to allow for threading a single query against multiple templates within reasonable time. We obtained better results than current comparable approaches, as well as significant reduction in execution time.

Acknowledgments

I would like to give my gratitude to all who gave me the possibility to fulfill this thesis. First of all, I wish to express my deepest sense of gratitude to my advisor Dr. Alioune Ngom for his guidance, encouragement, support and inspiring idea of this thesis. His instructions, comments and other assistances in preparing this thesis is invaluable. His influence pervades this thesis. I owe him lots of gratitude for having me shown the way of research.

I would like to thank to my committee members Dr. Majid Ahmadi, Dr. Luis Rueda and Dr. Jianguo Lu for their constructive criticism, valuable suggestions, technical assistance and referring through the course of this thesis.

My greatest appreciation is expressed to my family for their enduring love, encouragement, and support along the long way, without which I would be too weak to do anything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Bioinformatics derives knowledge from computer analysis of biological data. Fredj Tekaia at the Institut Pasteur offers this definition of bioinformatics: "The mathematical, statistical and computing methods that aim to solve biological problems using DNA and amino acid sequences and related information". Over recent years, bioinformatics has seen a substantial success in the field of genomic research. The success of genome-sequencing computation technologies in computational molecular biology has led to a large number of available sequences in the genome databases. The next field in the post-genome era is protein. Alisa Zapp Machalek, 2001 remarked in her NIH (National Institute of Health) Record [41] as "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. But proteins that twist into 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." On the other hand, biochemists have established that the spatial structure of protein determines its function, which includes protein's role in health and diseases. Thus knowing the structure of protein has large implications in understanding the protein's role in the body and to explore ways to control its action such as disease detection/drug design etc.

The Human Genome Project has led to the identification of over thirty thousand genes which may encode over 100,000 proteins as a result of alternative splicing. To understand the biological functions and functional mechanisms of these proteins, the knowledge of their 3-D structures is required. The Structural Genomic Initiatives, launched by NIH in 1999, intends to determine these protein-structures within a decade [12]. Unfortunately, the experimental determination of protein structures is not as easy as genome sequencing. The current laboratory techniques used to determine the 3D structure is x-ray crystallography and NMR spectroscopy. Both techniques are costly, time-consuming and difficult for high-throughput production.

Conversely, we have huge source of protein sequences from genomic database and the experimental methods to determine the protein sequence gives high throughput and is relatively cheap. The question is whether or not we can predict the three-dimensional structure of a protein based on its sequence. This is one of the most important and long-standing problems in computational molecular biology. The difficulty of determining the three dimensional structure of proteins has led to an increasing gap between the huge number of protein sequences and the limited number of protein structures. 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 [6]. Thus an affordable approach and a high throughput method is urgently needed in order to understand the biological systems and to shorten the gap of sequence and protein structures. Thus method can revolutionize in the field of proteomics.

Several approaches have been used to predict protein structures from sequences, with varying levels of success. Ab-initio methods initiate any means of calculating coordinates for a protein sequence without referencing existing protein structures. However, relatively little success has been seen in this approach [64]. The comparative or homology modeling method [38] attempts to find a structure based on the strength of sequence similarity to another protein of known structure. This is based on the premise that similar sequence implies similar structure. This approach relies on the success of existing known structures of

close homologue [31] and the subsequent accuracy of alignment.

A sequence can determine the three dimensional (3D) structure but some spatial constraints lead to the fact that a sequence may fold in different shapes. It depends on the spatial position of sequences that occur in the structure. This leads to the idea of predicting protein structure of a sequence by "inverse folding" [29] that is, to fit the query sequence into a known structure from the template database according to its spatial position and determine how best it fits. This method is called protein threading.

1.2 Challenges

The Structural Genomic Initiatives took the strategy to determine the protein structures using experimental methods only for a small fraction of all proteins and to employ computational techniques to model the structures for the rest of the proteins [12]. The basic premises behind this idea are that a limited number of unique folds in nature and different proteins share significant structural similarity. So, determining the unique structural folds in the laboratory may allow us to predict the vast majority of other proteins "within the modeling distance" of these proteins. Protein threading represents one of the most promising such techniques according to the report in CASP2 [17], CASP5 [31] and CAFASP3 [19].

The fact is that only 3% of the known sequence families include a member of known 3D structures [6] in the protein databases. Sensitive sequence profile methods can extend the range of comparative modeling to the point where 12% of the families can be assigned to a known 3D structural super family [31]. But in contrast, the probability of novel protein having a similar fold to a known structure is currently as high as 60% - 70% [30]. This observation motivates the approach of protein threading to be successful among other prediction methods.

Since the number of unique folds in nature is fairly small, a structural template database of several entries can be constructed by excluding those with highly similar structures. Thus, each unique fold represents one or several templates. Protein threading chooses the best fit template as a structural prediction for the query sequence through finding the optimal

alignment between the target sequence and each structural template in the database. This best-fit template becomes the basis on which a structural model for the target sequence can be built. If we can have a criterion to quantify such alignment, based on that an efficient algorithm can find the optimal alignment in affordable time.

The alignments between the sequence and the template include the criteria of local conformation consistency and spatial conformation consistency. The spatial conformation consistency is often modeled as pairwise contacts. That means, if two amino acids are spatially nearby in the structure, then the two amino acids in the sequence which are aligned to them in the structure should have strong pairwise contact potential. Biochemists consider the pairwise interaction is important for protein folding. On the other hand, insertion and deletion can happen anywhere in the loop or core part of the protein. But predefined core elements, gap restriction (i.e. insertion or deletion can happen only in loop region etc.) are some imposed assumptions to simplify the computation, but these hide the quality of threading alignments.

While it is true that secondary structure elements are more conserved than the loop regions, significant structural information is carried by the residues, that are in the loop regions. Insertions and deletions are also observed between similar proteins even inside corresponding secondary structure elements. Threading methods that can handle full alignments without arbitrary restriction of core elements will thus have an important advantage [71].

There are different approaches [11] [42] [22] [78] [71] [85] [88] [80] of protein threading algorithm based on the local or non-local interaction and variable-length-gap or gapless threading have been proposed. Many of above methods suffered with exhaustive searching, some of them compromise with quality. The underlying fact is that if pair-wise interaction and variable-length gaps are allowed in the alignment between the query sequence and the template structure, then the protein threading problem is NP-hard [35]. The quality of threading depends on the success of the scoring function and the efficiency depends on the searching strategy for optimal alignment.

1.3 Goal

In light of the preceding discussion, we seek a protein threading algorithm, which is capable of treating the pairwise contact potential and various length of gaps in the sequence-structure alignment. 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. The problem formulation is expected to overcome predefined core elements or gap restriction. Such an algorithm expects better alignment accuracy than those similar approaches in the literature review.

2. Efficiency: Such a threading algorithm should run fast to overcome the present limitation of selecting only a limited number of representative templates. Of course, it is difficult to outperform those algorithms that do not deal with pairwise contact potential or solve a limited version of the actual problem. The new method is expected to reduce large sequential computational time to thread each of the protein templates ($\simeq$ 1000) and terminate in a reasonable time.

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

1.4 Problem Statement

The Protein Threading Problem (PTP, for short) is to determine the three dimensional structure of a given but arbitrary protein sequence, called query, from a set of known structures called templates of other proteins.

Yadgari et al. [88] discusses a genetic algorithm approach for PTP, however their method threads a query against a single template only instead of a template database. Furthermore, their technique is very slow and therefore can not be used for a very large query and/ or template. They report results on short queries/templates with length of 300 amino acids at most.

As far as we know, the current threading algorithms only thread a single query against a single template. To thread against many templates, the given algorithm is applied sequentially many times, each time with a different template. Large queries cannot be threaded that way against a template database, within satisfactory and respectable time bounds, particularly where the templates are large. As a matter of fact, a vast number of proteins have never been attempted due to their sizes [89].

With an appropriate parallelization, there is a possibility to thread (very) large queries against a set of (very large) templates within reasonable time bounds. There is even a possibility to thread multiple queries against multiple templates within reasonable time bounds.

We propose two parallel approaches for the PTP problem. Both are parallelizations of our novel evolution strategy (ES) method for protein threading. The first method we call SQST-PEST (for Single Query Single Template- parallel Evolution Strategy threading) threads a single query against a single template. We use ES to find the best alignment between the query and the template, and ES is parallelized. The second method, we call SQMT-PEST (for Single Query Multiple Templates Parallel ES Threading) to allow for threading a single query against multiple templates within reasonable time.

1.5 Contribution and Results

The problem representation of protein threading used in Yadgari et. al. 2000 [88] dismisses the predefined core elements and gap restriction. We have used the exact formulation as Yadgari, 2000. The main contribution of this thesis is to propose an evolutionary strategy approach using the representation of Yadgari to the protein threading problem (which is NP-hard). The proposed ES threading performs better in quality of solution and in computation time than the method described in [88]. We also propose parallel architectures to reduce large sequential computation time for the protein-threading based structure prediction approach. We have shown that in practice, almost all instances of the intractable protein threading problem can be solved by our methods in affordable time. The main contribution of our

thesis can be outlined as follows:

- The evolutionary strategy approach for protein threading: The ES approach, we have used is in the class of $(\mu + \lambda)$-ES. It conducts effective searching of optimal alignment for protein threading that considers
  - Elitist selection and small changes (mutation) in alignment that cause significant difference in threading.
  - A novel technique of recombination and mutation, which is well suited and efficient for our threading problem
  - A design of normalized fitness criteria, based on energy function, which is distance-specific contact potential matrix [11] and it is a good candidate to evaluate fit of an offspring.
- Parallelization of the method: We proposed two different architectures for parallel approach and compared their performances. Both architectures reduce large sequential time.
  - It gives flexibility to choose a large number of templates from protein databases, that eventually increases the threading quality.
  - It allows to breed significant number of offsprings in each generation, that may contribute to converge in (near) optimal threading quickly.

1.6 Organization

The rest of this thesis is organized as follows. The chapter 2 describes the background of this research. At first it describes the basics about protein and different levels of protein structures. Different structure prediction methods and their success are introduced in section 2.2. Section 2.3 introduces the basic components of protein threading methods and its complexity. The

brief description of evolutionary strategy is given in section 2.4. Parallel computing and parallel architectures are introduced in section 4.2.

A survey on protein threading algorithm and related study have been described in 3. The detailed survey of protein threading algorithm and their problem formulation/ representation has been discussed. The evolutionary algorithm and parallel approach in protein threading has been investigated. The limitations of the existing methods are also outlined in this chapter.

Chapter 4 describes our proposed method for solving protein threading method in parallel using evolutionary strategy. Section 4.1 describes our basic evolutionary algorithms, proposed for protein threading. It discusses in details the problem representation, methods of using mutation and recombination as genetic operators, ES algorithm and its complexity. The parallel approaches are discussed in section 4.2. The two proposed methods of parallelizing to facilitate the large sequential computing time have been discussed with details of their algorithm.

The implementation and experimental results are discussed in 5. The implementation details of the energy function used, ES method and its parallel architectures are discussed in 5.1. The implementation environment and details of the system environment for the experiments are detailed in section 5.2. Section 5.3 describes about the source and details of the data set used for the experiments. The details of experiments, results and discussion are described in section 5.4.

The chapter 6 concludes the thesis. It narrates the description of the summary of work done. The limitations and future works are also outlined in this chapter.

Chapter 2

Background

Protein is a large molecule composed of one or more chains of amino acids in a specific order. Proteins are essential to the structure and function of all living cells and viruses. Many proteins are enzymes or subunits of enzymes. A protein chain folds into specific shape in order to function in its aqueous state. The conformation of protein can be parsed into several different levels ranging from local to overall spatial structure. There are several inter-atomic forces that play instrumental roles in the formation of protein shapes. In protein structure prediction, these factors are modeled in their scoring function. This chapter provides an overview of the architecture of protein structure, methods of protein-structure prediction and a survey of protein threading method which is more promising in the structure prediction era.

2.1 Protein Structure

The basic building blocks of proteins Proteins are amino acids. The spatial conformation of a protein is dominated by the order of the amino acids contained in it, and their side chain properties. Protein structures are described in four different levels: primary structure, secondary structure, tertiary structure and quaternary structure.

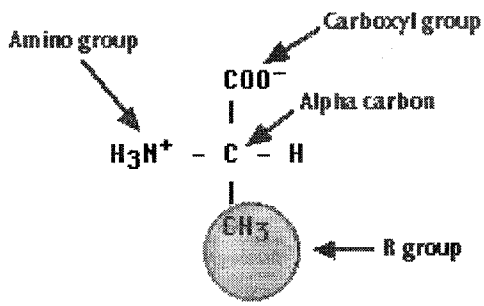

Figure 2.1: Structure of an amino acid (taken from [76]).

2.1.1 Amino Acids

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

Two amino acids are connected to form a peptide by a chemical reaction. Multiple amino acids can be connected sequentially to form a polypeptide as shown in 2.2.

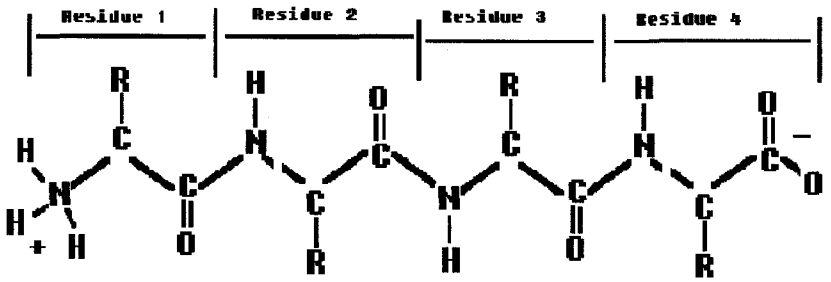

Figure 2.2: Sequence of amino-acids forms a polypeptide (taken from [76]).

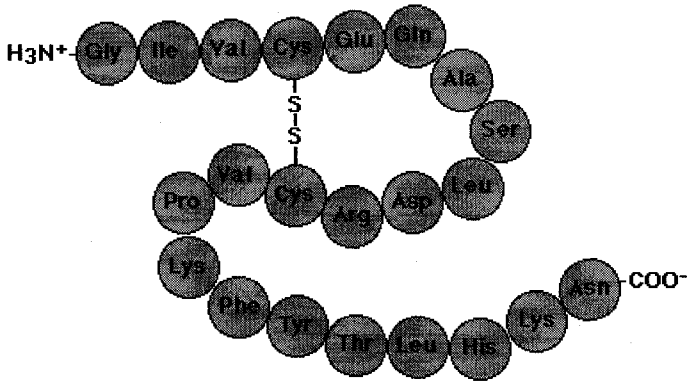

Figure 2.3: Primary structure of protein is simply the order of its amino acids (taken from [76])

Primary Structure

The primary structure of a protein describes the sequence level of protein and it represents the linear order of amino acids contained in it as shown in Figure 2.3. Although the primary sequence does not explicitly express any structure information about the protein molecule, the spatial conformation of proteins can be determined by their primary sequence [5]. Thus it is still fair to say that the three-dimensional structure of a protein can be determined by its primary sequence. Proteins fold up to complex shapes due to the bonds formed by the side chains. As the strong bonds play roles among two nearby residues in the sequence, the bonds between distant residues also contributes to form the fold. The former one is called local interaction and the latter ones are called non-local interaction. The interactions between residues are known as pairwise contacts.

Secondary Structure

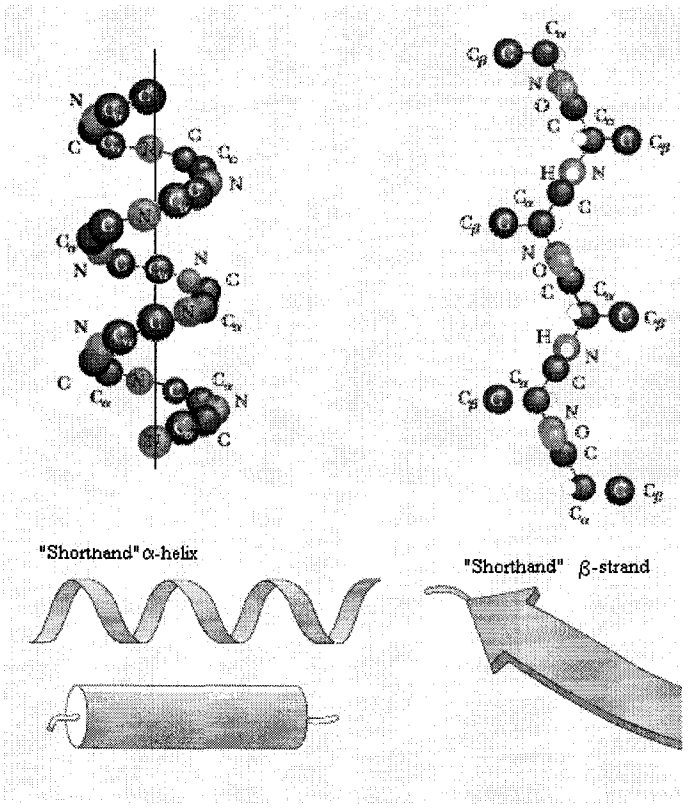

Figure 2.4: Secondary structures of a protein are alpha-helix and beta-sheets (taken from [76])

A segment of protein primary sequence can fold into a secondary structure because of the local interactions. The backbone of core structures in protein is known as secondary structure. α helix and β sheet are two common types of secondary structure and also known as core region. The core regions are more conserved and called template or fold. They are connected by loop. There are strong hydrogen bonds among the residues within one secondary structure, but the bonds between the residues within a loop are weak.

A combination of a few secondary structures that appears in several different proteins is called a motif. An example of a motif is the helix-loop-helix and such structure has clear role in protein function while some motifs has no specific role in the function of protein. A domain is a more complex combination of secondary structures that by definition has a very

specific function. Therefore it contains an active site, which is the a section of the protein where some binding to an external molecule can take place.

Tertiary Structure

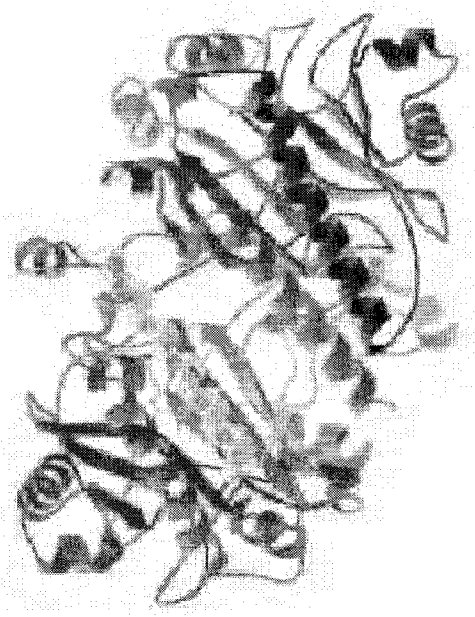

Figure 2.5: Tertiary structure of a protein is composed of connected secondary structures with loops (structure of protein (12as_a) taken from PDB)

A protein may have only one domain, or may contain several. All of them taken together form the protein's tertiary structure. Due to non-local interactions, all secondary structures in a protein can form a specific tertiary structure connecting each other with the loops and packed of side chains. It represents the 3-dimensional structure of a protein. Frequently, we use folds to denote the type of tertiary structure of a protein.

Quaternary Structure

Many large globular proteins consist of several polypeptide chains which are kept together by various forces such as hydrogen bonds or disulfide bonds. Such multiple-chain bonds

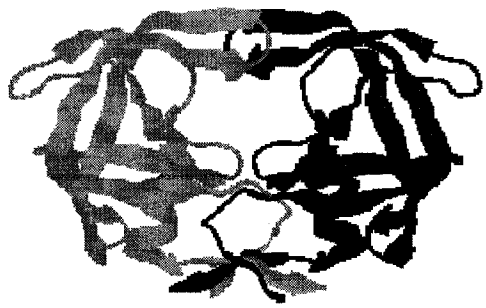

Figure 2.6: Quaternary structure of a protein is composed of multiple-chain bonds (taken from [76]).

represents the spatial relationship among all the protein chains and known as known as tertiary structure as shown in 2.6.

2.1.2 Factors Determining Protein Structures

The 3D structures of protein are much more restricted whereas the sequence varies extremely due to evolution. This happens because of the fact that the fraction of residue exchange 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 [56], if the proteins of length not less than 100 residues have the sequence identity of more than 35% then the two proteins may have very similar structure. In spite of a large number different proteins in nature, there are only fewer (approximately 1000) different protein structural folds [49], which is the basis of the success of protein threading method in protein structure prediction.

2.2 Structure Prediction Methods

2.2.1 Computational Methods

Several computational methods for protein-structure prediction have been developed and proposed. These methods can be grouped in four categories such as homology modeling, ab initio, fold recognition and consensus methods.

Homology Modeling

Homologous proteins are evolved from the same ancestor. So they have some degree of sequence and structural similarity. An underlying principle for homology modeling is that if a set of proteins are homologous, then their 3D structures are more conserved than their primary sequences [51]. Homologous modeling is suitable for those target sequences which are obviously being homologous with a known three-dimensional structure [38]. These kind of target sequences are called homology modeling targets. Homology modeling builds the tertiary structure of a target sequence by comparing the target sequence to all of its homologous sequences [4], recognizing the most conserved part through multiple alignments [70], copying coordinates for these conserved segments from one homolog with known structure and finally refining the whole structure through the energy minimization technique [75].

Ab-Initio Folding

The targets that do not have the same fold as some templates or do not have the homologous proteins with a known structure are referred to ab-initio folding method for structure prediction. This method builds the structural model directly from the target primary sequence alone. The scoring function used in this method are both based on traditional atomic force field [8] and knowledge based [64] used in recent years. This method starts from simulating the folding pathway of a protein and finally building the structural model of the sequence [15].

Currently there is no reliable and general scoring function that can always drive a search

to a native fold, and there is no reliable and general search method that can sample the conformation space adequately to guarantee a significant fraction of near-natives (less than 3.0 angstroems RMSD from the experimental structure). Some methods for ab initio prediction include Molecular Dynamics (MD) simulations of proteins, Monte Carlo (MC) simulations that do not use forces but rather compare energies, and Genetic Algorithms which try to improve on the sampling and the convergence of MC approaches.

Protein Folding

The protein folding problem is the following: Given the amino acid sequence of a protein, determine:

- where exactly all of its α-helices, β-sheets, and loops are, and
- how they arrange themselves in motifs and domains

The ability to determine a protein's native folded state is becoming critical with the massive increase in genetic sequence discovery and hence protein primary structure.

Protein Threading

The target sequences that do not have homologous templates but do have the same fold as some templates are referred to fold recognition or protein threading. The 3D-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 best fit template as the target sequence [4]. After the best-fit target is selected, the structural model of the sequence is built on the sequence based on the alignment with the chosen template [10] [63] [95].

Consensus Method

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-server. The first generation meta-servers such as Pcons [40] did not exploit the information provided by the individual servers. But the recent meta-servers such as 3DSX [18] and PMODX [33] can now employ fragment assembly techniques to construct a new consensus model from their inputs, which is an impressive and promising advance.

2.2.2 Success of Methods in Prediction

In [57], use of evolutionary information in the form of multiple-sequence alignments results 70.8% accuracy in secondary-structure prediction of globular proteins. The four membrane protein drops it to 70.2%. The strand residues are predicted at 65% accuracy.

At third round of Critical Assessment of Protein-Structure Prediction (CASP3), Jones et al. 1995 did best with 77% of residues correctly predicted. But the corresponding number for the subset of difficult targets is 73% [20]. The limitation is due to the fact that these methods are based on sequence-specific information in the close vicinity of the residue which considered only local interaction. The secondary structure is also dependent on the non-local interactions [13].

A recent approach uses profiles made by position-specific scoring matrices as input and output predicts three consecutive residues simultaneously. This output expansion and unique balloting method's overall secondary structure prediction performance is 77.2% - 80.2% (77.9% - 80.6% mean per chain) [52]. With respect to blind prediction, this work is preliminary and awaits evaluation by CASP4 [94].

2.3 Protein Threading

Protein sequences that have the same fold as some proteins with known 3D structures but do not have homologous proteins with known structures are suitable for protein threading [14]. Protein threading makes a structure prediction through aligning the residues of the target sequence onto the positions of each template structure from a set of templates. It determines

whether the target can have the same fold as the template or not. For the optimal alignment, gaps are allowed to the some extent and a pair-wise scoring function can be used. Thus it does not consider that all sequence residues are aligned to a template position and all template positions are aligned by a sequence residue [68]. The two major criteria for a successful protein-threading application is the design of energy function to measure the quality of alignment and an efficient alignment algorithm to search optimal alignment.

Protein threading matches the query sequence onto a known protein structure. The compatibility of a sequence to that structure can be estimated based on this matching. The query sequence is a linear structure that is a sequence of amino acid. Protein structure possesses the back bone of protein such as core (alpha-helix, beta-sheet), loops including three dimensional coordinates of each amino-acid atoms in the space. Assigning a structure to a query sequence needs to thread the sequence through all known templates (representative subset of known proteins), estimate the compatibility and find the most compatible structure. It requires some important components such as the query sequence of the protein, representative template database derived from Protein databases [6], an alignment algorithm and an objective function to evaluate the quality of an alignment of the sequence onto a given template structure (as shown in Fig 2.7).

2.3.1 Components of Protein Threading

The following four steps aggregates the protein threading approach for protein structure prediction. The protein threading components have been shown in the Figure 2.7 (taken from http://www.bcbio.de/zib_lecture/ Prakt2004/lect/may_threading.ppt and modified). The success and efficiency of the protein threading depends on these steps.

Template Database

A representative set of protein structures extracted from the PDB database is used to form the template database. Template is consist of only core segments removing the loop. This

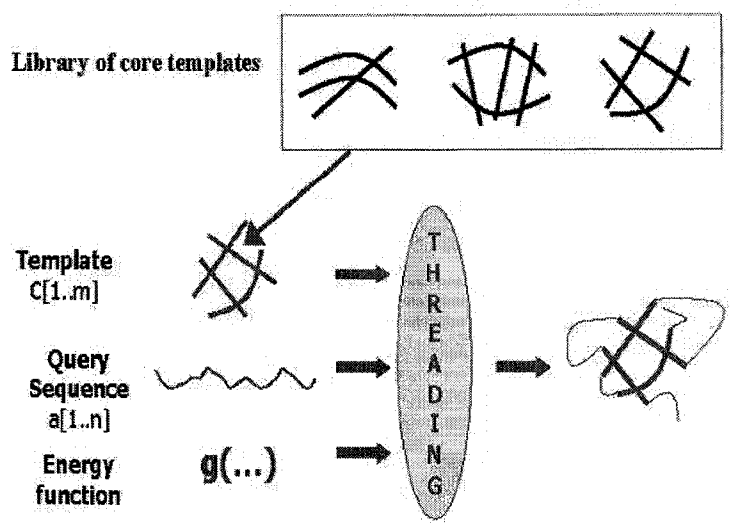

Figure 2.7: Protein threading process and basic components

generally involves selecting protein structures from databases such as SCOP [48], or CATH [50], after removing protein structures with high sequence similarities. Fold library in 1999 contains 600 to 900 entries depending on the similarity threshold applied [13].

Threading Alignment

The simplest alignment is no-gap alignment in which the query sequence is mounted over an equally-long part of a target fold. It is not used for structure prediction, but used for testing and adjusting the scoring functions [29]. To approach protein structure prediction, a sequence fragment is fitted over a part of target fold allowing gaps both in sequence and fold [69].

The alignment algorithm depends on the scoring function used. Methods based on non pair-wise scoring functions uses conventional [29] and double dynamic-programming algorithms [69] to find the optimal alignment. Various approaches convert the scoring function to non pair-wise and then apply a dynamic-programming algorithm [22] [78]. Others apply direct algorithm such as branch-and-bound searching algorithm [37] or Monte-Carlo sampling algorithm [9] or linear programming [91] [84] have been investigated. All these approaches are discussed in more details in section 3.1.

Scoring Function

The scoring function measures the fitness between target sequences and templates based on the knowledge of the known relationships between structures and the sequences. A good scoring function should contain mutation potential, environment-fitness potential, pair-wise potential, secondary-structure compatibilities and gap penalties [87] [95] [11]. The quality of the score function is closely related to the prediction accuracy.

The pair-wise interaction is an important measure on the quality of protein threading method. Thus two approaches in protein threading has been observed where the pairwise interaction is explicitly counted in alignment algorithm and a substitution energy score like PAM250, BLOSUM62 matrices are used to score them. In other methods, alignment is further assessed by the pair-wise interaction score. Some softwares are used to generate the output for certain template considering all related potentials such as PROSA, a software tool for the analysis of 3D structures of proteins based on [65] [28], FROST (Fold Recognition Oriented Search Tool) based on [45] ROSETTA [55] .

We used the distance dependent pair-wise energy matrices proposed by Bryant and Lawrence 1993 [11] and is further detailed in section 5.1.1.

Ranking and Modeling

In this phase of threading method, select the threading alignment that is statistically most probable as the threading prediction. The ranking for structural prediction is widely acceptable using z-score [10] [66] and machine learning [84].

Fold Recognition using Z-score: The z-score is defined as 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. The proposed method in [10] calculates z-score after threading a pair of sequence and alignment to cancel out the composition bias. Let the z-score is denoted by Z_{raw}. An accurate Z_{raw} can cancel out the sequence composition bias and offset the mismatch between the sequence size and the template length. But this

randomization and to calculate accurate Z_{raw} is time consuming. The steps are generally taken as follows:

1. Find the optimal alignment between the target sequence and template, and determine the optimal alignment score, E_{opt}
2. Shuffle the query sequence randomly
3. Calculate the alignment scores based on the existing random-alignment, without searching the optimal alignment again.
4. Repeat step 2-3 N times (N is on the order of several thousands). The mean energy score E_{avg} and standard deviation, σ_E is obtained from the N alignment-scores.

The Z_{raw} can be calculated using the following formula in Eq. 2.1 (adopted from [10]):

$$Z_{raw} = \frac{E_{opt} - E_{avg}}{\sigma_E} \tag{2.1}$$

Skolnick and Kihara, 2001 argued that the use of pair potentials improves the fold specificity [66]. The mean Z-score of correctly identified template structure can examine the hypothesis as a function of the various potentials used. In this method, they thread the query sequence onto each template in the template database and determine each optimal energy. Thus, $\langle E \rangle$ and σ being the mean and standard deviation values of the optimal energy in all templates of the structural database. If the Z-score for the Kth structure is having the energy E_K, then it can be determine by the equation 2.2 (adopted from [66]) as follows:

$$Z_{raw} = \frac{E_K - \langle E \rangle}{\sigma} \tag{2.2}$$

This is one measure of the effectiveness of scoring function. Because, they did not randomize the sequence in the evaluation of Eq. 2.2. However, sequence randomization is a computationally expensive process, and it would be a significant advantage to be able to avoid it, especially when threading is done on a genomic scale.

The accuracy of the predicted structure can be assessed by examining the predicted side-chain contact-maps [66]. In this method, the number of correctly predicted contacts, N_c are recorded. Generating random alignments of the query sequence in the correct template structure can do the measure of significance of this quantity. In general, threading with single sequence, other than to a template database has no significance on the value of N_c. To address this issue, [66] suggests the following metric: if the average number of correctly predicted contacts is N^0 and standard deviation is σ^0 for the best alignments of the query-sequence and each of the template-structure in the template database, then the Z-score for the number of correctly predicted contacts can be determined using following Eq. 2.3 (adopted from [66])

$$Z_{con} = \frac{(N_c) - N^0}{\sigma^0} \tag{2.3}$$

This quantity measures the significance of a given number of predicted contacts.

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.

2.3.2 Computational Complexity

If pair-wise interaction is considered in the scoring function and variable gaps are allowed in the sequence-structure alignment, then the problem is NP-hard [35] [2] [91]. The proof is given in [35] and [36] [37] proposed the optimal solution using special-purposed branch-and-bound method. In [2], Akutsu and Miyano conducted a comprehensive study concerning the approximation of the protein-threading problem. They demonstrated that the threading problem is MAX SNP-hard. It means that no approximation algorithm can guarantee an accurate solution within polynomial time. They have shown that the protein-threading problem is much harder to approximate than preserving approximation of reduction from the DENSE-k-SUBGRAPH problem, for which only an $O(n^{-0.3885})$ approximation algorithm is

known so far. They also proposed an $O(|E|)$ approximation algorithm for this problem where E is the number of pairwise contacts in the template and gave a constant-factor approximation algorithm when the templates have a planar contact map graph [2]. However, not many templates have a planar contact graph [81].

2.3.3 Energy Function

The energy function for protein threading is generally trained by using a database of known protein structures (i.e. training database). An accurate representation of the free energy represents the best-possible scoring function if the thermodynamic hypothesis are correct. Such scoring functions are commonly referred to as energy functions or contact potentials.

Optimization method for energy function is based on formulation of the fact that protein in its native structure has the lowest energy compared with other random structures. Here native structure is meant as the original structure of the protein in the nature. Based on this thermodynamic hypothesis, the first method was developed in [44] and assumed that protein sequence in each non-native conformation have energy higher than the energy of the native conformation by some margin. Thus a set of linear inequalities have been imposed for this requirement and the standard linear programming techniques have been imposed for the optimal contact potentials.

The empirical scoring function is a potential-of-mean-force approach where the distribution of the interactions obeys the Boltzman distribution [11]. They compute the contact potential as the logarithm of the ratio of the frequency of contacts observed in the training database over the frequency of contacts expected in the reference state. Based on probabilistic description of the threading model, the scoring function has been developed in terms of structural environment states [7].

Other approaches used maximizing the z-scores. The z-score is the measure of the distance in standard deviations of a sample from the mean. In [46], they maximized the average of individual z-scores to obtain optimal potential for a set of proteins. Thus the proteins with

low z-scores dominates the averaging procedure.

Many other approaches have been investigated. As examples, the hydrophobic contact potential of Huang et al., 1996 [25] reflects packing in the hydrophobic core using only two residue classes, hydrophobic and polar, and is remarkable for its explanatory power given its simplicity and near absence of adjustable parameters. Maiorov and Crippen 1994 [43] used linear programming to enforce a constraint that the native threading scores lower than others, but such approaches tend to be brittle. Bryant and Lawrence 1993 [11] used logistic regression, based on multidimensional statistics. Boltzmann statistics is the foundation of many threading methods such as [28]. White, Muchnik et al. 1994 [77] derived a formal probability model based on Markov Random Fields.

2.4 Evolution Strategy

In this section, we briefly describe an optimization technique called evolution strategy (ES). The ES is the main optimization tool used in our optimization problem.

Rechenberg [54] pioneered ES and Schwefel [61] introduced ES as method to solve optimization problems. ES belongs to the class of evolutionary algorithm that solves the problem as a method of natural selection. It works on an encoded representation of the solution. Each candidate solution (individual) produced by the problem representation is called as chromosome. In early ES, the individuals are encoded as vectors of real number and was proposed as an optimization method for real valued vectors. But recent ES is open for any encoding. The pool of candidate solutions created in each generation is called the population. Each solution is associated with fitness value. The fitness value represents the performance of the individual solution in relation to the parameter being optimized. It also represents an individual solution's performance in respect to other potential solutions in the search space. ES is a random guided hill-climbing technique in which all candidate solutions are produced by applying mutations on each parent individual. The best solutions generated in one generation becomes the parent for the next generation. ES is an iterative method and the process of selection,

reproduction is repeated until some termination criteria is reached. When the termination criteria is reached, the solution to the problem is represented by the best individual so far in all generations. The basic steps of an ES algorithm can be summarized as follows:

1. Generate an initial population of λ individuals
2. Evaluate each individual according to fitness function
3. Select μ best individuals as parents for the next generation
4. Apply reproduction operator i. e. mutation on μ and create λ offsprings where $\lambda > \mu$
5. Go to step 2 until a desired solution has been found or predetermined number of generations have been produced and evaluated.

Rechenberg's ES was developed with selection, mutation and a population of size one, while Schwefel 1981 introduced recombination and population with more than one individual, and provided a nice comparison of ESs with more traditional optimization techniques. In the standard recombinative ES, pairs of parents produces offsprings via recombination, which are further perturbed via mutation. Two common variations of ES introduced by Schwefel [61] are the (μ, λ)-ES and $(\mu + \lambda)$-ES. These two approaches differ in the selection of individuals for the next generation. In $(\mu + \lambda)$-ES, μ best individuals are selected from the set of λ offsprings to form the parents for the next generation, while in $(\mu + \lambda)$-ES, μ best individuals are selected as parents for the next generation from all $(\mu + \lambda)$ individuals.

The most common variants of evolutionary algorithms are Evolutionary Programming (EP) [21], Evolutionary Strategy (ES) [54], Genetic Algorithm (GA) [24], and Genetic Programming (GP) [32]. All these approaches differ in three respects such as representation scheme, reproduction operators, and the selection methods.

2.5 Parallel Computing

Parallel processing has made tremendous impact on many areas of computer application. With the raw computing power of parallel computers, it is possible to address many applications that were beyond the capability of sequential computing techniques [74]. Evolutionary algorithm is a natural paradigm to map to a multi-processor computer [16]. Parallel approaches are are more demanding where the problem space is very large and fitness function is expensive for criteria evaluation.

2.5.1 Parallel Architecture

There are a variety of parameters that can classify and measure the performance of the parallel computing architecture. Processor interconnection is one of the most important issues in which processors of the interconnected computer exchange information and it affects on the efficiency of performance. The two extreme alternatives for processor interconnection have been compared in [39] as shared memory and distributed memory as follows:

In shared memory, one processor can communicate with another by writing the information into a global shared memory location and having the second processor read directly from that location. This makes inter-processor communication very easy, but introduces problems having to do with simultaneous access of a unique memory location by multiple processors. Thus, it tends to have fewer processors than their distributed memory counterparts.

In distributed memory architecture, each interconnected processors has their own local memory. Processors are connected on a communication network and share information by passing it through this network. If communication network is complex then it may reduce the performance of communication, but it allows many processors to be interconnected. Massively parallel machines prefer to have a distributed memory.

In [39], he compared the two architectures and concluded that advanced routing techniques have made the cost of communication between any two processors roughly the same as in one processor. He also mentioned that the network of workstations on a common Ethernet sub-

network has no notion of processor neighborhood and is likely where parallel optimization algorithms will find the most widespread use.

2.5.2 Message Passing in Parallel Architecture

Message Passing Interface (MPI) is a paradigm used widely on distributed memory architecture of parallel machines. The goal of MPI is to develop a widely used standard for writing message-passing programs. As such the interface establishes a practically portable, efficient, and flexible standard for message passing [67] [23]. MPI offers an application-programming interface not necessarily for compilers or a system implementation library. It allows efficient communication and at the same time allows for implementations that can be used in heterogeneous environment.

SHARCNet used distributed memory architecture and MPI can be used to communicate among multiple processors.

2.5.3 Parallel Implementations in Bioinformatics

Certain methods for analyzing genetic/ protein data has been found to be extremely computationally intensive, providing motivation for the use of powerful computers. Parallel algorithms have been well implemented in many areas of bioinformatics. Yap et. el 1995 applied parallel algorithms on biological sequence analysis, especially on multiple sequence alignment problem using speculative computation [92] [93]. Schmidt et. el. used parallel algorithms on protein database searching [60]. Bokhari et. el. applied parallel techniques to implement the dynamic programming approach in DNA sequence alignment [62]. Xiao et. el. applied parallel algorithm on a cluster of workstation for gene clustering [79]. Zhang et. el. used a data level parallel algorithm for structure prediction [96]. Thomas and Amato used Standard Adaptive Template Parallel Library to parallelize their method for protein folding problem [72]. Akutsu used parallel approach to search similar structure using structure-structure alignment in parallel computer [?]. More recently, we found the marriage of mathematical programming

and parallel computing for solving protein threading problem in [90]. Yanev used linear programming LP and network flow formulation to solve protein threading problem in parallel processors using commercial LP optimizer, CPLEX.

Chapter 3

Related Study

This chapter discusses the protein threading algorithms that are currently available and gives a brief survey on the existing methods that are relevant to our problem.

3.1 Survey on Protein Threading Algorithm

There are several protein threading algorithm based on different assumptions. The two basic considerations that determine the complexity of this algorithm are whether or not (1) variable-length gaps are admitted into the alignment, and (2) interactions between neighboring amino acids from the sequence are considered into the score function. If variable-length gaps are not permitted [44] then alignments are restricted to substructures of equal length as from the database and will be partially out of hydrophobic registration.

Johnson et. al, 1992 [69] permitted variable lengths but pair-wise interactions are ignored and only the local environment is considered. In [58], they evaluated interactions with respect to the structure's original native sequence in stead of the sequence actually being threaded. In these cases, the global optimum threading can be found using dynamic programming alignment method. Dynamic programming employs an affine gap penalty and it biases the search to prefer loop lengths in the model structure's original sequence. This make more difficult to recognize for distant structural homolog if their loop length differ substantially.

In addition ignoring amino acid interactions means giving up a potentially rich source of structural information [59].

Among the approaches, Lathrop and Smith, 1994 gave the first mathematical formulation of the problem considering both pair-wise interaction and variable-length gaps which applies "special purpose" branch and bound algorithm for searching optimal alignment [36] [37]. The method guarantees to find the optimal solution but it has turned out as exhaustive-searching and time-consuming procedure for larger sequences. But the success of [36] persuaded other researchers to consider the success of prediction and and it is still an active area of research. For this reason current search algorithms adopt any of two choices: (1) Approximation Algorithm: It may find the optimal solution in many cases and very good solution in respect to others, but sometimes fail to find the optimal (2) Exact Algorithm: It may terminate rapidly in many cases, but sometimes must require an exponential amount of time.

3.1.1 Exact Algorithm

Branch-and-bound algorithm:

Lathrop and smith [36] [37] have proposed a branch-and-bound method to solve the threading problem. [36]is the first approach to formulate the protein threading problem and optimized special purpose branch-and-bound method. They considered pairwise interaction removing the loops between cores. But in [37], they considered gap explicitly between the cores, but not in the core segment. In this approach pairwise contacts is temporarily ignored in subspaces, uses dynamic programming algorithm for searching subspaces and then estimate the bound of the objective function in the entire space. This process is repeated and some subspaces are discarded based on the estimated bound. It has been reported that this method guarantees the optimal threading but approximately 5% to 10% threading instances of their own test sets can not be handled within a reasonable time [37].

Protein threading is based on the fact that certain proteins with similar structure have different loops and similar cores. So the idea is to predict protein structure from the query

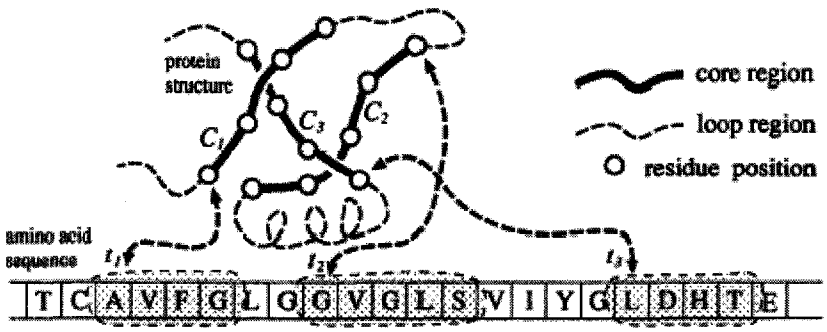

Figure 3.1: Protein Threading Problem (adopted from [ATM97] Page 4)

sequence in a way so that the query sequence fits to a known structure considering its spatial position and determine how best it can fit. The protein threading has been shown in the Figure 3.1 (adopted from 3.1 Page 3)

The structural model is determined by replacing amino acids with place holder in a known structure. Place holders are kept associated with certain properties of amino acids. The structural model keeps record of spatial constraints of the structures such as distance to the other amino acids how much inside or outside the whole structure is. Then this structural model of known structure is aligned with the query sequence to determine the best fit structure.

So threading procedure relies on two major components among others is:

(1) An alignment algorithm to position a sequence to a structure.

(2) Score function to evaluate the conformation as which one fits best.

The basic inputs for protein threading problem are:

1. A query protein sequence, A with n amino acids such as $a_1 a_2 ... a_n$.

2. A core structural model C, with m core segments c_j

   - The length of c_j, each core segment.
   - Core segment c_j and c_{j+1} are connected by loop region, j and for each j maximum length ($l_j{}^{max}$) and minimum length ($l_i{}^{min}$) bound.
   - The local structural environments ($\alpha - helix, \beta - sheet$ or loop) for each amino acid position in the model.

3. An efficient score function to evaluate a given threading.

The output for the threading is:

A set T = $t_1, t_2, ...t_m$ of integer such that the value of t_i indicates that what amino acid from sequence, A occupies the first position from each core segment, c_i.

Thus threading is an alignment between the sequence and the core structural model allowing the above conditions. The score function and gap between cores play the vital role and if pair-wise alignment is allowed on variable length gaps the problem becomes NP-hard [35].

Basically [36] [37] gave the formal presentation of the protein-threading problem and they proposed branch-and-bound algorithm, first exact solution for finding the global optimal threading using pair-wise interaction of amino-acids and allowing variable length gaps. According to that formulation the basic concept of constraints, loop length bounds and scoring functions are as follows:

Spacing, order and interval constraints (adopted from [36] Page 368):

$1 + \sum_{j<i}(c_j + l_j^{min}) \leq t_i \leq n + 1 - \sum_{j \geq i}(c_j + l_j^{min})$

$t_i + c_i + l_i^{min} \leq t_{i+1} \leq t_i + c_i + l_i^{max}$

$b_i <= t_i <= e_i$

With t as some threading, the scoring function is (adopted from [LS94] Page 368):

$f(t) = \sum_i g_1(i, t_i) + \sum_i \sum_{j>i} g_2(i, j, t_i, t_j)$

where given inputs g_1, g_2 are contributions from individual core elements and pair-wise core elements respectively.

value t_i determines the position of amino acids from sequence A in core segment i.

In the protein threading problem, an amino acid sequence and a set of 3D protein structures are given to find an optimal alignment between spatial positions of a 3D structure and amino acids of a sequence minimizing the score of a suitable energy function. [36] [37] defines the problem in a formal way and this formal definition is modified in a simpler form [2] without loss of generality as follows as Figure 3.1 (adapted from [2] Page 4).

The formulation described below is (adopted from [2], Page 4) as follows:

Input: sequence $s = s_1 s_2 .....s_n$ over a fixed alphabets representing amino acids

core lengths: $c_1, c_2, ..., c_m$

score function: $g(i, j, t_i, t_j)$ such that $(g(i, j, t_i, t_j) \geq 0)$

Output: m-tuple t = $(t_1, t_2, ..., t_m)$ which maximizes a total score such that

score (t) = $\sum_{i<j} g(i, j, t_i, t_j)$

under the condition that $1 \leq t_1, t_i + c_i <= t_{i+1}, t_m + c_m \leq n + 1$

By inverting the sign of values of score functions and by adding a constant factor, the problem is converted to maximization problem for applying approximation algorithms. In [LS94], two kinds of score functions $g_1(i, t_i)$ and $g_2(i, j, t_i, t_j)$ are used and they are generalized as $g(i, i + 1, t_i, t_{i+1}) = g_1(i, t_i) + g_2(i, i + 1, t_i, t_{i+1})$ Here they ignore the computing time of $g(i, j, t_i, t_j)$ since it can be computed in polynomial time for most scoring function. Also the effect of loop regions is taken into account by adding a length of loop region to the length of a core region and by modifying $g(i, j, t_i, t_j)$ suitably. Thus the loss of generality has been taken care of.

Divide-and conquer method:

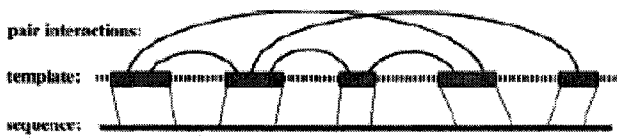

Figure 3.2: A schematic of sequence-structure alignment. The line at the bottom shows the target sequence. Each box represents a core secondary structure (α-helix or β-strand) of the template. The dotted lines between boxes represent the loop regions. An arc between two core secondary structures indicates that there exists at least one pairwise interaction between the two cores. The two lines between a core and the target sequence represent a gapless alignment between the core and the sequence. (adopted from [85] Page 344)

In the PROSPECT [86] [85] [80] structure prediction program, Xu et. al. developed a divide-and-conquer algorithm based on the observation that if the cutoff distance of the pairwise contacts is fixed to 7 $\dot{A}$, then three-quarters of the template contact graphs are topologically simple.

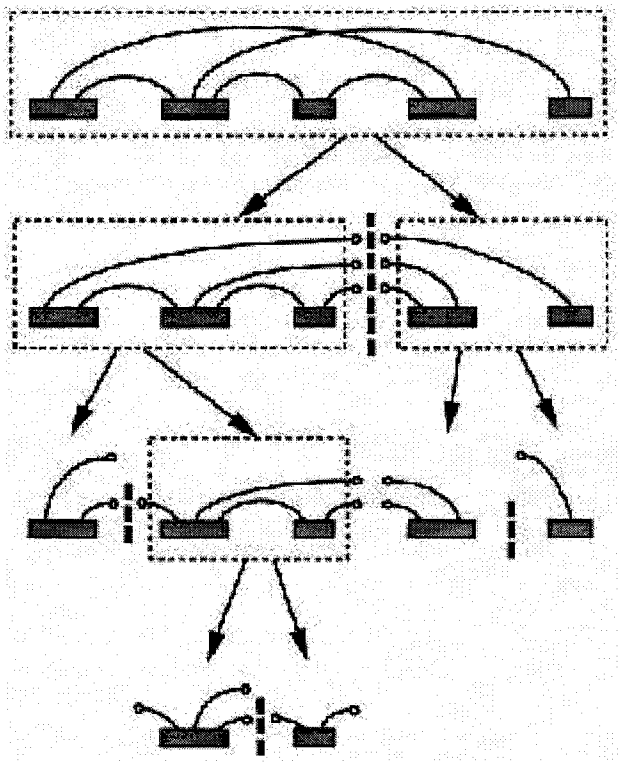

Figure 3.3: Partition of a template structure that forms a tree structure as indicated by the arrow. The first row shows the template with five core elements. The second row shows a partition of the template into two substructures, one with three cores and the other with two cores. A broken arc ended with a circle is called an open link. Third and fourth shows further partitions. (adopted from [85] Page 345)

In this method, they assumed that the contact potentials can be calculated only between residues of core secondary structures (α-helix or β-strands). They also assume that alignment gaps are confined to loop regions. Based on that assumptions, the threading problem can be schematically represented in Figure 3.2. The goal is to find an alignment between the template and the target sequence so that total energy score is minimized.

In PROSPECT [86], an optimal alignment between the target and each template is determined in two steps:

1. Finding an optimal alignment between the target and core elements of the template, while penalizing length differences between the corresponding loop regions, using a divide-and-conquer algorithm [85]

2. Alignment loop regions are done separately after core elements are aligned, using a sequence-sequence alignment algorithm like Smith-Waterman.

The quality of an alignment between a target and a template's core elements is measured by a linear combination of 1. singleton fitness that specifies secondary structure and solvent accessibility 2. Pairwise contact term the specifies how preferable of two residues in close contact 3. Term which penalizes the length differences of corresponding loops.

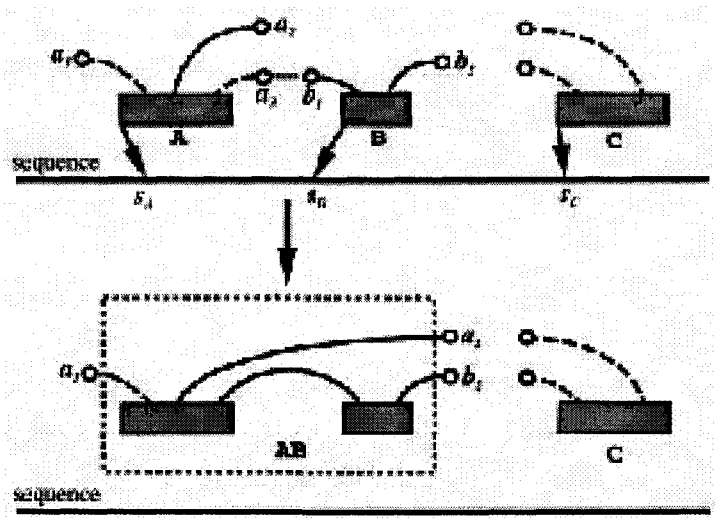

Figure 3.4: A schematic example of divide and conquer algorithm. (adopted from [85] Page 345)

The threading employs a divide-and-conquer strategy to solve the optimal threading problem. For this purpose, the algorithm pre-process the template by repeatedly dividing it into substructures until each substructure contains only one core secondary structure. Dividing the template cuts an interaction between two cores into two open links, represented as shown in Figure 3.3. The algorithm solves the entire optimal alignment problem by recursively solving a series of sub-alignment problems between sub-structures and sub-sequences, under various constraints, and combining these sub-alignments in a consistent way. Figure 3.4 illustrates the idea, using an example from the last partition step in Figure 3.3.

The idea is to split a template into two subsegments such that each segment is connected to as few external cores as possible, to recursively align each subsegment to the sequence

respectively, and finally to merge the alignments of two segments to form an alignment for the whole segment. For topologically complex contact graphs ($\geq$ 400 residues) of about one quarter it runs very slow.

To deal with situations where corresponding cores of two aligned structures may have different lengths, PROSPECT allows deletion/additions of up to two residues at each end of a core in the template. The number of deleted/added residues from/to a core is determined by the optimization algorithm and it increases the computational time significantly.

3.1.2 Approximation Algorithm

Recursive Dynamic programming:

This algorithm repeatedly uses a dynamic programming algorithm to match the target sequence to the template [71]. At each iteration, the local alignment between the target sequences and the templates is searched by dynamic programming algorithm. A segment of the target sequence is fixed onto a segment of the template if a significant similarity is achieved. The iteration is repeated until no significant similarity is found. The unmatched segments of the target sequence are interpreted as gaps.

RDP optimizes the following scoring function for evaluating a threading alignment f of the target sequence A with a known protein structure B:

$$\phi(f,,B) = \gamma*\phi^S(f,A,B)+\delta*\phi^C(f,A,B)+\varepsilon*\phi^H(f,A,B)+\zeta*\phi^P(f,A,B)-GAP(f,A,B)$$

In this scoring function ϕ^S scores the alignment f with respect to the alignment f with respect to well-known sequence based mutation matrices, contact capacity potential ϕ^C is the compatibility of the query sequence with the structural fingerprint of the template. ϕH is a scoring preference of an amino acid to a specific structural position with respect to hydrophobicity and solvent exposure. ϕP denotes the pair interaction term of the potential. GAP penalizes insertions and deletions. γ, δ, ε, ζ are weighing parameters, which have been calibrated empirically.

RDP algorithm is described in pseudo code in [71]. The steps are as follows:

- Detect local region of high similarity among the target and template sequence
- Local alignment
- Exploit sequence as well as structural signals
- Any pair of locally aligned segments divides the unmatched region of both protein into two parts.
- They can be processed independently with the same approach.
- After dividing, the changed structural features of the template are recorded.

The algorithm proceeds recursively, until in the local alignment step, no more significant similar segment pairs are found e. g. only one core structure.

Interaction-Frozen Approximation:

The interaction-frozen method was first used to solve protein threading by [22]. In this approach, one end of each contact is fixed to the residue in any current alignment position including initial alignment. Then an iterative step is used to search for the next alignment based on frozen pairwise potential [22]. [78] used the same approach, but improves the iterative step with dynamic programming algorithm until the algorithm converges. No optimal alignment is guaranteed.

Monte-Carlo Sampling and Simulated Annealing:

[11] used the simulated annealing method to solve the protein threading problem. Optimal alignment between the target sequence and the template has been searched using Simulated Annealing algorithm. [42] [47] initializes one initial sequence-structure alignment and then used the Monte carlo sampling technique to generate the next alignment from the current alignment. Both approaches usually take enough computational time to converge to the optimal alignment and it is unaffordable to thread for long sequence. The method allows

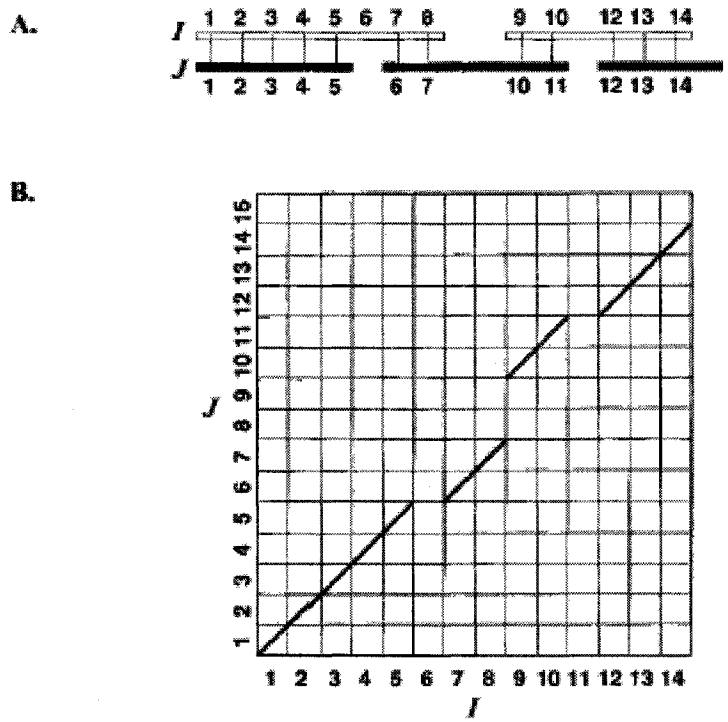

Figure 3.5: Representation of protein-protein alignment. A, example of alignment, B, its matrix representation (adopted from [47] Page 523)

gaps and insertion both in the query sequence and in the template structure. The alignment representation is defined as follows:

An alignment between two proteins of length I and J is represented by a matrix A_{ij} where $i = 1, .., I$ and $j = 1, .., J$:

A_{ij} = 1 if i is aligned with j

= 0 otherwise

Another way of representing an alignment is by a pointer p_i

p_i = j if is aligned to j

= 0 if i is not aligned to any residue

The representation did not allow double matches (i, e, $\sum_{i=1} A_{ij} \leq 1$). The reverse of any fragment in the alignment is also forbidden, i.e. if $A_{ij} = 1$, then for any $i' > 1$ and $j' < j$ $\rightarrow A_{i'j} = 0$. Under this constraints, matrix A_{ij} should have the form shown in Figure 3.5, i.e. an alignment composed of runs of aligned residues separated by gaps in either or in both

proteins. These runs are referred to below as fragments of alignment. Each fragment is a set of matches ($A_{ij} = 1$ for $(i,j) = (i',j'),(i'+1,j'+1),...,(i'+L,j'+L)$) framed into gaps $A_{i-1,j-1} = A_{i+L+1,j+L+1} = 0$. These fragments are used in the move set as building blocks.

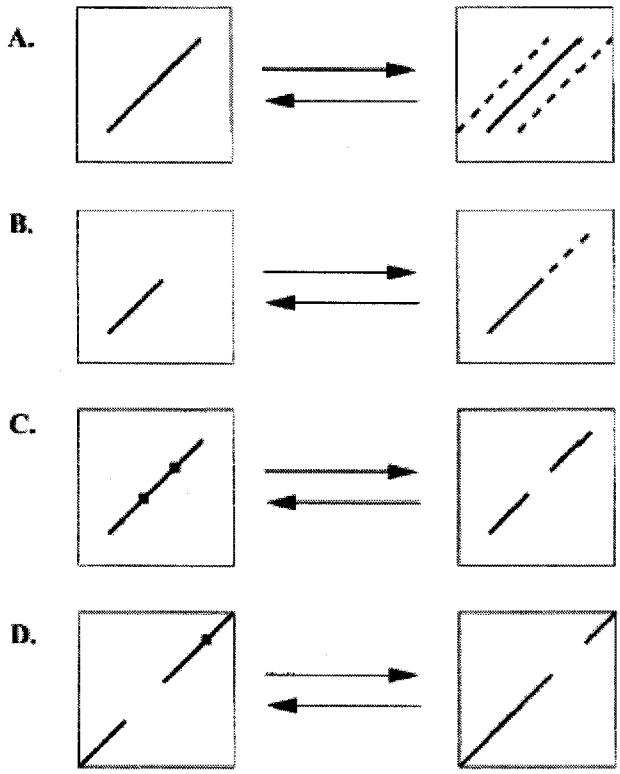

Figure 3.6: Move set of Monte-carlo method, A, Shift, B, Shrink/ expand C, Split/ merge, D, Jump (adopted from [47] Page 523)

Each move in the move set is designed to change the alignment preserving most in the matches and hence, leading to small change in energy. It also allows easy introduction of constraints on the minimum length of a fragment or maximum length of a gap. This flexibility is achieved by making moves on fragments, rather creating and destroying single matches. The fragment length is constrained to be greater or equal to $L_{min} = 6$ residues. The moves are shown in Figure 3.6.

To sample possible sequence-structure alignments and search for the alignment with minimal energy, they used the Monte Carlo (MC) method. The power of MC procedure is that

it allows us to find a global minimum on a variety of rough landscapes. In the search for a minimum, it samples possible alignments and allows to study statistical properties of the energy landscape.

Linear Programming:

Mixed Integer Programming for RAPTOR

RAPTOR [83] used a linear programming based approach for protein-structure prediction via threading. The protein threading problem is formulated as a large scale integer programming based on the contact map graph (shown in Figure 3.7) of the protein 3D structure template. RAPTOR is based on the definition and alignment model of threading shown in [84] [81] [82]. They followed the basic assumptions which are widely adopted by threading community [11] [36] [42].

In their work [84], the threading score function is well defined and consists of singleton score such as the environment fitness score E_s, mutation score E_m, secondary structure compatibility score E_{ss} and the gap penalty score E_g, pairwise interaction score E_p. So the scoring function, E can be derived as follows (the equation is adopted from [84], Page 4):

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

where $W_m, W_s, W_p, W_g, W_{ss}$ are weight factors determined by training.

Global alignment and local alignment methods are employed to align the sequence to the template. In order to reduce the too many variables in the the formulation, they simplify the template contact graph by merging all the vertices representing the residues in one core into a single vertex. This is based on the assumption that no gaps are allowed in the core. So they model each core as a vertex and adding one edge between two cores if there is at least one residue-residue contact between them. It is explained in the Figure 3.7 (adopted from [81] Page 5).

Then they construct a bipartite graph to describe all potential alignments between any core and any sequence position. They denotes D[i] as all the valid query sequence positions

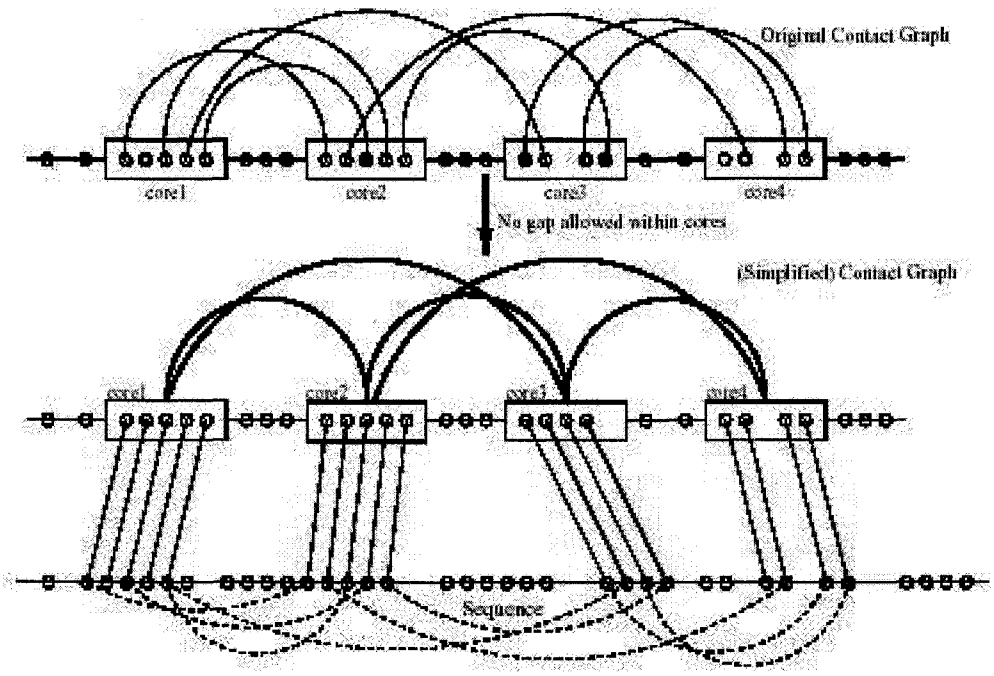

Figure 3.7: "A template contact graph and an example of an alignment between one template and one sequence. A small circle represents one residue. The solid arc in the original contact graph indicates that its two end residues have an interaction. A dashed arc shows that if two sequence residues having an interaction to each other, then the interaction score of these 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 residues which are aligned to two interacted template residues" [83] (adopted from [83] Page 5).

that c_i can be aligned to. R[i, j, l] denote all the valid alignment positions of c_j given that c_i is aligned to s_l. Figure 3.8 (adopted from [83] page 7) illustrates the example of D[i] and R[i, j, l].

They presented three versions of linear mixed-integer program formulations with three objective functions to formulate the sequence-template alignment problem and proved the third constraint set, CS3 [84] is the strongest when the integrality constraints on x and y variables are relaxed to allow real values between 0 and 1.

This method focuses on the formulation of the threading problem both in designing scoring function and efficient algorithm for alignment. The experimental result proves their quality and efficiency in formulation of RAPTOR [84].

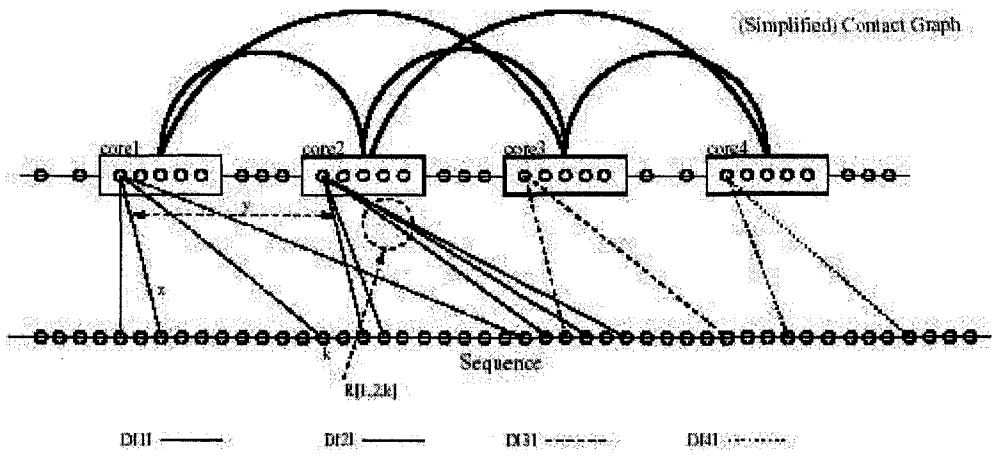

Figure 3.8: "Example of D[i] and R[i, j, l]. R[1, 2, k] is the set of potential alignment positions of core 2 given core 1 is aligned to sequence position k. Core 1 has five residues which have to be aligned to the sequence based on assumptions in the "Alignment Model" subsection. Thus, the first two candidate alignment positions of core 2 are invalid if core 1 is aligned to position k in order to avoid overlap". (adopted from [83] page 7)

In implementation of the RAPTOR, IBM OSL [26] package is used to optimize this MIP formulation. The above package was used to relax the integer program by allowing all x and y to be real between 0 and 1 and solve the resulting linear program. If the solution of the linear program is integral, then the optimal solution is found. Otherwise one non-integral variable is selected according to some criterion, and generate two sub problems by setting it to 0 and 1 respectively. These two subproblems are solved recursively.

For weight training, the weight factors are through optimizing the overall alignment accuracy and an SVM (Support Vector Machine) method is used to carry out the fold recognition. The Z-score is approximated by fixing the alignment positions, shuffling the query sequence randomly and calculating the alignment scores based on the existing alignment. Then the SVM-Light software is employed to adjust the approximate score.

Network-Flows and MIP

[91] has used the formulation of [36] and derived the objective function for linear programming formulation. But in the formulation of objective function, the presence of pair-wise interaction

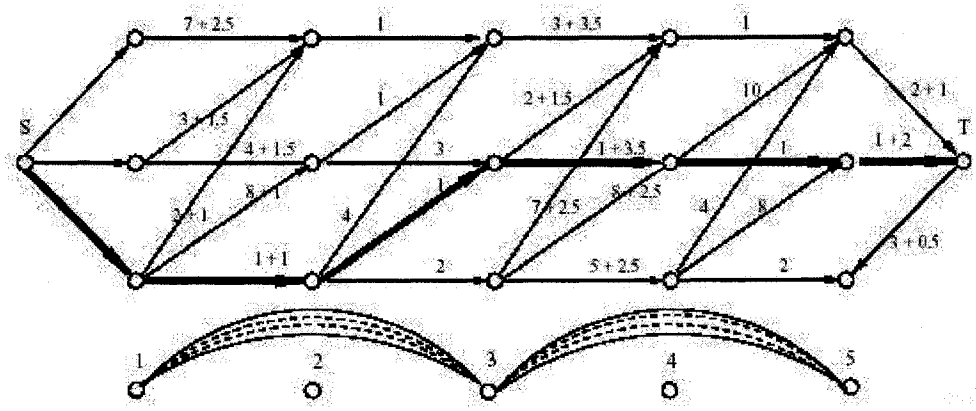

Figure 3.9: This graph corresponds to the network flow formulation of the problem. (adopted from [YA02] page 9)

between the segments contributes the non-linear term. In order to linearize the problem, they introduced new linear variables $z_{ijkl} \in (0,1)$ in the objective function in stead of non linear term, $x_{ij}x_{kl}$ and add some more constraints as described in [91]. Their formulation aims to be solved with commercial LP solver, CPLEX [27]. But their initial formulation fails because of weakness of LP-bounds formulation.

[91] introduces the network flows for the above formulation to solve in shortest path from an artificial source node to destination which is optimal alignment. The arc weights, c_e are related to the scores from the above original formulation and each weight is a sum of three numbers: segment to position cost, gap cost between segments and local interaction (if any).

The graph represented in Figure 3.9 (adopted from [91] page 9) gives more geometric insight for the problem of optimal aligning of some sequence with a core of 5 segments, each one with three possible placements. The path given in a thick lines has a length 5 but taking into account the pair-wise interactions (in this case (1,1,3,2), (3,2,5,2) - the path passes through the vertices (1,1), (3,2) and (5,2)). The costs for passing through these vertices ($c_{1132} + c_{3252}$) is added to 5 and obtain the actual length 14 (= 2+7+5) of the threading path. In the figure 5.1, lb is the bound obtained by relaxation and is calculated as [34], opt is the optimal value determined by computing the exact cost of the path. Thus if weights to all arcs and a table of the scores for the designated non-local pair-wise interactions are given, the

optimization problem will convert to find a path from S to T with minimal updated length.

The CPLEX was run on the MIP model generated from this network flows constraints on a large subset of instances. The results of LP relaxation attains optimal. But the fact is that all these properties are scoring dependent and they could be lost once the scoring scheme changed [91]. They used the CPLEX branching strategy to improve the LP-bounds by imposing branching on the SOS (Special order set) constraints in stead of on a single variable [27], but at the expense of adding extra constraints. The split and conquer algorithm is applied to split the problem into sub-problems and passing the best objective function value as a cutoff for the subsequent sub-problems. Thus by having the chance to start with the sub-problem which contains optimal path, all other sub-problems will be aborted by the LP solver at the moment when dual objective reaches the cut off value.

In [91], the experimental result is compared with the result of [37] and shows that in order to generate optimal threading for longer protein-sequence, the time limit varies between 30 min to 2 hour as compared as the same instances for [37]. But in [91] formulation they sacrificed the quality of the comparison with lower accuracy for the chance to test the long protein-sequence instances which were never attempted before.

3.2 EA in Protein Threading

The evolutionary algorithm has been applied on protein folding problem [73] [53]. We also found an evolutionary algorithm based approach for protein threading problem, known as genetic threading, proposed by Yadgari 2000 in [88]. The method concentrates on the development of suitable algorithm for (near) optimal sequence-structure alignment in protein threading approach based on genetic algorithm. This is the only research that we found in literature review in our knowledge so far which is related to protein threading and based on evolutionary algorithm. We took it as the key approach for our basis of research and we will outline their approach in brief description.

The algorithm puts the residues of query sequence onto the place holder of a known

protein-template and looks for a reliable alignment with the minimal free energy. This is a process of threading approach in protein structure prediction. Yadgari, 2000 used one of the available energy function, described in [11] for their threading algorithm. The method includes a representation of the problem, the suitable genetic operators and their approach for insertion/ deletion of residues in sequence-structure alignment.

Problem Representation: The method proposed a fixed length string to represent the individual solutions and to use a string of integers o, 1, 2, ..., K to represent the solutions where K is the length of structure. Each '1' in the string represents a residue from the sequence is aligned onto that position of the structure while each '0' represents no alignment of query-residue. Numbers bigger than 1 represent the number of query-residues that does not have suitable match on the next structural position. Each such solution validates the length of the representative string equals the length of the structure while sum of all integers in the string is equal to the length of the sequence. The representation is further described in details in 4.1.1.

Genetic Operators: The proposed method used genetic operators such as mutation and crossovers. The mutation was done by introducing a gap or by deleting the mismatched residues from the sequence. It was performed by increasing or decreasing randomly the integer in the solution string and offsetting the same amount in other positions. The proposed crossover represents a combination of two alignments. It performs by choosing a random position and building two new offsprings by concatenating the prefix of one with the suffix of other and vice versa. Since the arbitrary crossover of two alignments are not guaranteed to represent a valid alignment, each string is further validated.

The method did not use the predefined core elements or gap restriction for any representative solution. It used the trie data-structure to efficiently remove the duplicate solutions so that early convergence can be avoided. They reported good results for their threading method.

3.3 Parallel Approaches in Protein Threading

Akutsu and Sim, 1999 has developed a protein threading system based on multiple structure alignment [3]. They used a parallel architecture for comparing two similar structure. In similar structure search, an input structure is compared with all structures (several thousands of structures) in PDB. For that purpose, they used a simple master-slave model. The master process watches the status of all slave processes. If the master process finds an idle slave process, then it sends a protein structure, which is not yet compared, to the slave process. The slave process computes a structure alignment (using stralign [1]) between that structure and the input structure, and then it returns the result to the master process. Although this model is very simple, it works quite well because each comparison can be made independently. They reported that they could achieve near linear speedup ratio per slave process (up to 50 processes) by means of storing all 3D data in main memory.

Yanev et. al. 2003 [89] modified their linear-programming based approach used in [91] for parallelizing the computational algorithm in multiple processor using the capability of LP solver CPLEX. In this formulation, they used the properties of the MIP model that permits a decomposition of the main problem into a large number of subproblems (tasks). They showed a branch and cut technology can be efficiently applied for solving these tasks in a parallel manner and it leads to a significant reduction in the total running time. If the split and conquer method can split the problem into subproblems of equal size, then the intervals should be of equal length.

They applied the CPLEX call-back function technique [27] to make the task atomic and this method overcomes the slowing down of the global optimization process by learning from hardest one. The operations of the slave processor are as follows: (1) sending to the master the locally computed solution i.e. only the objective function value (2) receiving the record from the master, and (3) using it to update the cutoff value. This periodical updating of the logical record with the best global value allows the parallel processes to evolve simultaneously much faster. Furthermore, a cancelation of non-promising tasks in due course leads to significant

reduction in the total time.

Yanev et. al. has further improved their algorithm to super linear speed up and reported in [90]. In [90], they proposed a naive parallelization based on centralized dynamic load balancing since the amount of job is not known prior to execution. The splitting strategy suggests to split the problem in r subproblems, which are considered as tasks that need to be spread over p processors. The solution of the original problem is the minimum of all solutions computed in the manner. In the centralized dynamic load balancing, the tasks are handed out from a centralized location (pool) in a dynamic way. The pool is managed by master processor. The master processor sends the task to each slave processor from the pool. Once the slave processor is done, it sends back the result to master. The master processor keeps track of slaves and sends the tasks on demand to idle slaves.

3.4 Limitation of the Existing Methods

The several methods have been proposed for protein threading. But due to the complexity of the problem, most of the researches solve only limited version of actual problem.

While it is true that secondary structure elements are more conserved than the loop regions, significant structural information is carried by the residues, that are in the loop regions. Insertions and deletions are also observed between similar proteins even inside corresponding secondary structure elements. Threading methods that can handle full alignments without arbitrary restriction of core elements will thus have an important advantage [71].

Many of above methods suffered with exhaustive searching, some of them compromise with quality. Due to the computational bound, branch-and-bound and divide-and-conquer algorithms are limited to thread only small sequence and templates (< 400 residues). Similarly linear programming based approach considers resultant core energy in stead of actual residue-residue contact for simplicity of the problem.

[88] discusses a genetic algorithm approach for PTP, however their method threads a query against a single template only instead of a template database. Furthermore, their

technique is very slow and therefore can not be used for very large query and/ or template. They report results on short queries/ templates with length of 300 amino acids at most.

Since threading a query to each template is still a computational challenge, the current threading algorithms only thread a single query against a single template. In practice, we need to thread a query sequence onto multiple number of template-structures. To thread against many templates, the given algorithm is applied sequentially many times, each time with a different template. Large quarries cannot be threaded that way against a template database, within satisfactory and respectable time bounds, particularly where the templates are large. In matter of fact, vast number of proteins have never been attempted due to their sizes [89].

The parallel method we have observed in [3] concentrates in structure-structure alignment in order to see the similarity between two protein templates. They used this parallel approach to facilitate a profile-based search in threading. The method in [89] [90] observed substantial speed up in their linear-programming based approach for large proteins using a optimization software, but the method is limited to thread a single query onto a single structure.

Chapter 4

Proposed Method

In this chapter we described our proposed method for solving protein threading problem. Our method concentrate only on the algorithmic part of protein threading problem. We have demonstrated a new method based on evolutionary strategy applied on a suitable problem representation of Yadgari et. el. 2000 [88]. The problem representation, fitness function, mutation and recombination operators used, and ES algorithm has been discussed in details. In this thesis we also applied parallel computation on our evolutionary strategy based method.

4.1 Proposed ES Approach

4.1.1 Problem Representation

Our problem representation is same as in Yadgari [88]. Given a query Q and a template T, we seek to find the best alignment between Q and T. Given a template database $D = T_1, T_2, ..., T_N$, we seek to find the best alignment from the set of best alignments between Q and all T_i $(1 \leq i \leq N)$. The template that yields the best alignment is therefore, the solution, we are looking for. Thus finding the best alignment between Q and all templates in D gives the threading solution, and the search space is the space of all possible alignments. In order to apply ES to the protein threading problem, we therefore, need to represent an alignment

appropriately for ES to evolve it and find the best solution.

An alignment between Q and given T is represented as a fixed length string of integers $S = S_1, S_2, S_3, ..., S_{|T|}$ where $0 \leq S_i \leq |Q|, 1 \leq i \leq |T|$, and $\sum_{i=1}^{i=|T|} S_i = |Q|$.

Fig 4.1 shows the correspondence between a query-template alignment and an integer string representation of the alignment. $S_i = 0$ represents a structure deletion; $S_i = 1$ represents a match between a given amino-acid in the query and a template position (a match means that the amino acid is assigned to that position in the template's structure); $S_i > 1$ represents gaps in the sequence when aligned to template.

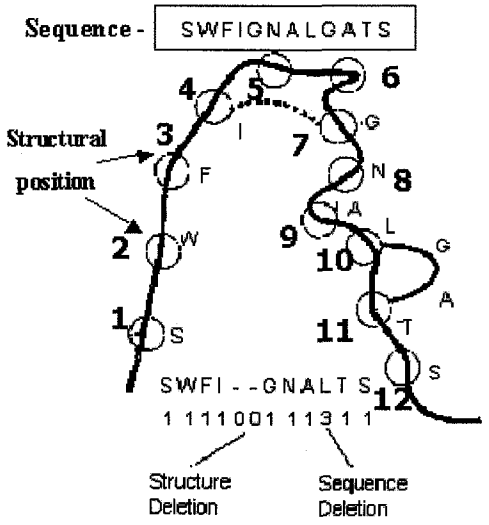

Figure 4.1: Representation of the problem formulation and schematic view of the threading process

The schematic view of the threading process is shown in Fig 4.1. The template's structure is presented by the bold trace through the circles that represent the structural positions (12 in the example). The query sequence on the top is threaded through this structure and the associate coding 111100111311 is shown in the bottom. In this threading, the first 4 amino acids S, W, F, I of the query are matched to the first 4 structural positions of the template (that is S, W, F, I assume the position 1, 2, 3 and 4). Next in the threading, the amino-acid G, N, A assume the positions 7, 8, 9 respectively, which are then set to values 1 in the encoding.

Position 5 and 6 of the template are not matched to the query (this is represented by the dash line) and thus specify the structural details in the alignment (thus, corresponding positions in the encoding are set to 0). The following amino-acid in the query, L takes position 10 of the template, but its corresponding position in the encoding is set to 3 (not 1) to signify that after this structural position, the next 2 amino-acids of the query are not matched to any structural position (this is an example of sequence deletion where the deleted letters are G and A). Thus a position $S_i = v$ in the encoding specifies that the next $v - 1$ letters of the query are not matched to any structural position while the current amino-acid takes position i. Last query letter, T and S are matched to last positions and corresponding positions in encoding takes 1 each. Below, we show the alignment between Q and T, and the corresponding encoding E.

T:	1	2	3	4	5	6	7	8	9	10			11	12
Q:	S	W	F	I			G	N	A	L	G	A	T	S
E:	1	1	1	1	0	0	1	1	1	3			1	1

In this paper, we use $(\mu + \lambda) - ES$ as our optimization method. Initially, we generate a random population of μ parents, that is each parent is initially a random integer vector, $S = S_1, S_2....S_{|T|}$, where $0 \leq S_i \leq |Q|$, $1 \leq i \leq |T|$ and such that the constraint $\sum_{i=1}^{i=|T|} S_i = |Q|$ is satisfied.

4.1.2 Fitness Function

Besides an appropriate problem representation, the design of our appropriate objective function is also very fundamental to the process of ES. The objective function is needed to assess how good or bad a candidate solution is. Candidates are selected according to their objective values and the best μ candidates among the current candidates are always selected as the new parents for the next generation. In a protein molecule, the bonds between the atoms of its amino acids determine the three dimensional conformation of the molecule in space. Thus the three dimensional structure (or the fold) of the problem is decided by the linear sequence of its amino acids. A protein always assumes a stable fold, and such fold has always

the lowest possible energy state. Therefore, an energy function can be used to determine, if a query protein, such that when some of its amino acids match some positions of the template protein, it folds at the lowest possible energy state. The energy state of the template's fold is known as well as its atomic coordinates in three dimension. The query's actual fold is unknown but we want to predict it. When the query is aligned to the template (as discussed earlier) and folded accordingly, we can then compute the energy of the fold. If the actual unknown fold of the query is the same as template's fold, then it will have the lowest energy state (which is the energy of the template's fold), assuming that the fold is attained from the best alignment between the query and the template.

Many energy functions are defined in literature. In this paper, we use the energy function discussed in [88] which is itself based on a more complex energy function studied in [11]. Given an alignment, its total energy (that the energy of its associated fold) is

$E_{total} = E_{single} + E_{pair} + E_{gap}$,

where E_{single} and E_{pair} are computed from the energy matrices of [11], and E_{gap} is the alignment gap penalty function which is set to 3 energy units. E_{total} is a function of the amino-acid type, the distance between amino-acids, the hydrophobicity of the amino-acids and the alignment gaps. E_{single} describes how well the individual amino acids of the query match their assigned structural positions of the template. E_{pair} reflects the pairwise interactions between the amino-acids. We refer the reader to paper [11] for discussion of how E_{total} is computed from the energy matrices.

The best solution vectors are those whose corresponding alignments or associated folds have the lowest energy. Since our ES maximizes an objective function, we transformed energy function appropriately for ES to maximize. The transformed energy function is the objective function that ES will use in order to select the best solution. We will use the term fitness function as the energy value is normalized such that it is between 0 and 1.

Given a query Q, a template T and a solution vector $S = S_1, S_2....S_{|T|}$, our fitness function

is defined as

$$F(S) = 1 - \frac{E(S) - E_{min}}{E_{max} - E_{min}}$$

where $E(S)$ is the actual energy of the fold associated with the alignment corresponding to S, E_{min} (respectively E_{max}) is the lowest (respectively highest) bound on the energy value and can be obtained by assigning to each column of the alignment, the minimum (respectively maximum) possible energy value from the energy matrices. Thus we always have $E_{min} \leq E(S) \leq E_{max}$. The closer $E(S)$ is to E_{min} then the lower is the energy of the fold, and thus the larger is the fitness of S.

4.1.3 Mutation

Mutation helps evolutionary process to explore new areas of the search space by generating totally new solution vectors. It also helps to maintain the diversity of a population, which in turn helps to avoid getting stuck in local optimum solution. Mutation operation randomly alter certain positions of a given solution vector. With our representation, mutation should be designed in such a way that it always produces valid solutions, that is the constraints $\sum_{i=1}^{i=|T|} S_i = |Q|$ and $0 \leq S_i \leq |Q|$ must be satisfied on resulting solutions. Our mutation operation gives valid vectors and is as follows:

We randomly generate an integer $n\epsilon[0, \frac{|T|}{2}]$ and randomly select two positions $p_1, p_2\epsilon[1, |T|]$, $p_1 \neq p_2$, to alter. We then increase S_{P_1} and decrease S_{P_2} by a same small random amount $m\epsilon[0, \frac{|Q|}{4}]$, given a parent vector S to produce a mutant vector. The offset m must be selected such that $0 \leq S_{p_i} \pm m \leq |Q|$, to ensure the validity of the mutant. This process of altering a pair of positions is repeated n times. For example, if parent $S = 1110401031$, $n = 1$, $p_1 = 4$, $p_2 = 9$ and $m = 2$ then after mutation , the mutant will be 1112401011. It should be noted that mutation operation is applied λ times on the current set of parents to yield λ mutants.

4.1.4 Recombination

We also applied a recombination operator on randomly selected pairs of parents. Recombination aims to combine genetic materials from two parents and pass them on to the next generation, depending on the fitness of the parents. First, we generate a random integer $n\epsilon[0, \mu]$ as the number of recombination operations to apply on the current population of μ parents. We then repeat the process of recombination n times, as follows:

Randomly select two parents S_1 and S_2, randomly generate a binary string M (we call 'mask') and construct the offsprings C_1 and C_2 in the following manner: at position p, $C_{1p} \leftarrow S_{1p}$ and $C_{2p} \leftarrow S_{2p}$ if $M_p = 1$, else $C_{1p} \leftarrow S_{2p}$ and $C_{2p} \leftarrow S_{1p}$ if $M_p = 0$. The following example shows the recombination process.

S_1	=	1	1	0	2	0	1	0	3
S_2	=	2	0	1	0	3	1	0	1
M	=	1	0	0	1	0	1	1	0
C_1	=	1	0	1	2	3	1	0	1
C_2	=	2	1	0	0	0	1	0	3

We may need to correct a child C_i, if it is not valid, so as to satisfy the constraints discussed earlier. In the example, C_1 and C_2 are invalid and thus will be corrected to make them valid.

4.1.5 ES Approach for Protein Threading

Figure 4.2 illustrates our ES approach for protein threading, called EST algorithm. The basic iteration of EST is shown in Figure 4.3. Starting from a random initial population U of μ candidate solution, we create subsequent generations by recombining members of U and then create a set of M of λ mutatants from the set of R of recombinants, and finally, the next μ parents are selected from the best in R and M. We also keep track of the best solution so far and preserve it across generations. The inner while-loop locally optimizes the current best solution in $(R \bigcup M)$ in order to escape a local optimum trap, and the best solution so far is updated only if it is weaker than the current best in R and M. Our method is elitist since

- Given Q and T, create random population $U = \{U_1, U_2, ..., U_\mu\}$
- Evaluate (U)
- $Best\_so\_far \leftarrow$ best solution in U
- Repeat
  - $R = \{R_1, R_2...., R_\mu\} \leftarrow$ Recombine (U)
  - $M = \{M_1, M_2, ..., M_\lambda\} \leftarrow$ Mutate (U)
  - Evaluate $(R \bigcup M)$
  - Current $\leftarrow$ best solution in $R \bigcup M$
  - $i \leftarrow 0$
  - While $F(current) \leq F(Best\_so\_far)$ and $i \leq 10$ Do
    - $C \leftarrow$ Mutate $(Current)$
    - $B \leftarrow$ Mutate $(Best\_so\_far)$
    - Current $\leftarrow$ best among C, B and $Current$
    - $i \leftarrow i + 1$
  - If $F(Current) > F(Best\_so\_far)$ Then
    - $Best\_so\_far \leftarrow Current$
  - $U = \{U_1, U_2, ..., U_\mu\} \leftarrow$ best solutions in $R \bigcup M \bigcup \{Current, Best\_so\_far\}$
- Until stopping criteria attained
- Return $Best\_so\_far$

Figure 4.2: EST Algorithm.

the best solution in a current generation and the best solution so far are passed on the next generation.

4.2 Parallel ES for Protein Threading

We propose two parallel evolution strategies for protein threading. The Single Query Single Template Parallel ES Threading (SQST-PEST) method threads one query against one template. The Single Query Multiple Templates Parallel ES Threading (SQMT-PEST) method threads one query against a set of templates. Both parallel approaches are implemented on

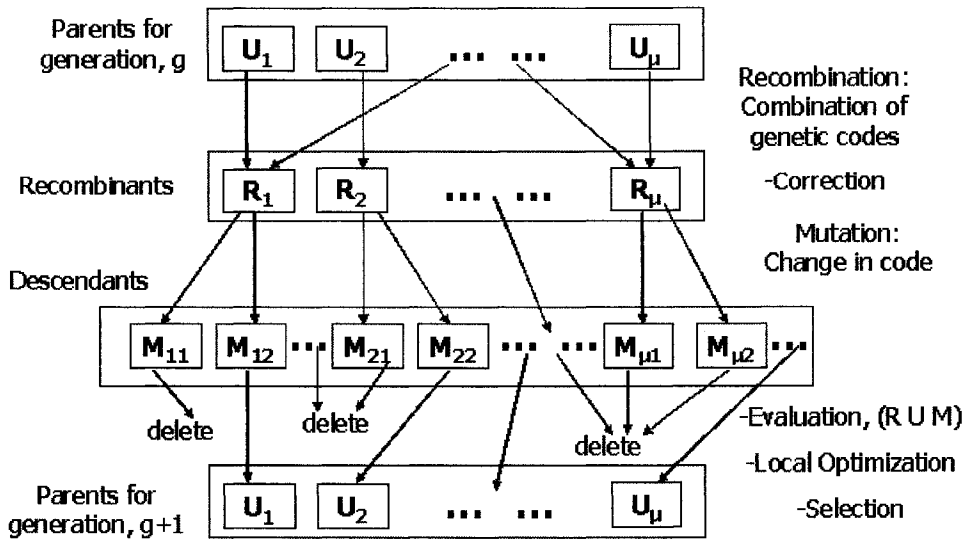

Figure 4.3: Iteration of EST

grids architecture. The parallelization are based on master-slave architectures, that is, one processor, the master, is selected to distribute tasks among other processors, the slaves, which process their given tasks independently of each other. The master collects results from slaves and returns the best threading solution. The following sections discuss SQST-PEST and SQMT-PEST algorithms.

4.2.1 SQST-PEST Approach

Figure 4.4 shows the SQST-PEST algorithm for threading one query Q against one template T. Figure 4.5 visualizes the flow chart of SQST-PEST approach.

Given a query Q and a template T, the master processor creates an initial population $U = \{U_1, ..., U_\mu\}$ randomly. The master recombines and mutates U to produce a population of R recombinants and M mutants, and then distribute the set $R \bigcup M$ among p slave processors. Each slave, $S_j (1 \leq j \leq p)$ receives the global best-solution so far, along with its subset P_j from the master. S_j evaluates each solution in P_j, locally optimizes the current best solution in P_j, and returns its best solution so far to the master. The master collects each slave's

1. Master: Given Q and T
   - Create random population $U = \{U_1, ..., U_\mu\}$
   - $Best\_so\_far \leftarrow$ Best solution in U
   - Repeat
     - $R = \{R_1, ..., R_\mu\} \leftarrow$ Recombination (U)
     - $M = \{M_1, ..., M_\lambda\} \leftarrow$ Mutate (U)
     - Split $R \bigcup M$ into p sets $P_1, ...P_p$
     - Send $Best\_so\_far$ and set P_j to slave S_j, $1 \leq j \leq p$
     - Receive $Best\_so\_far_j$, $F(P_j)$ from S_j
     - $Best\_so\_far \leftarrow$ Best of all $Best\_so\_far_j, 1 \leq j \leq p$
     - $U = \{U_1, ..., U_\mu\} \leftarrow$ Best in $\bigcup_{j=1}^{j=p} P_j \cup \bigcup_{j=1}^{j=p} \{Best\_so\_far_j\}$
   - Until stopping criteria reached
   - Return $Best\_so\_far$
2. Slave S_j: Given $P_j = \{U_1, ..U_{\frac{\mu+\lambda}{p}}\}$ and $Best\_so\_far$
   - Evaluate (P_j)
   - $Current_j \leftarrow$ Best in P_j
   - $i \leftarrow 0$
   - While $F(Current_j) \leq F(Best\_so\_far_j)$ and $i \leq 10$ Do
     - $C_j \leftarrow$ Mutate$(Current_j)$
     - $B_j \leftarrow$ Mutate$(Best_so_far_j)$
     - $Current_j \leftarrow$ Best among $C_j, B_j, Current_j$
     - i = i+1
   - If $F(Current_j) > F(Best\_so\_far_j)$ Then
     - $Best\_so\_far_j \leftarrow Current_j$
   - Send $Best\_so\_far_j$, $F(P_j)$ to Master

Figure 4.4: SQST-PEST Algorithm

best solution and the energy and fitness values of elements in P_j, called $F(P_j)$. The master appropriately updates the global best-solution so far, and then selects the best μ solutions from $R \bigcup M$ and each slave's result to create a new set of parents $U = \{U_1, ..., U_\mu\}$. The master repeats the process of recombination, mutation and so on until our stopping criteria is attained. The stopping criteria is either a convergence (that is the global best solution is

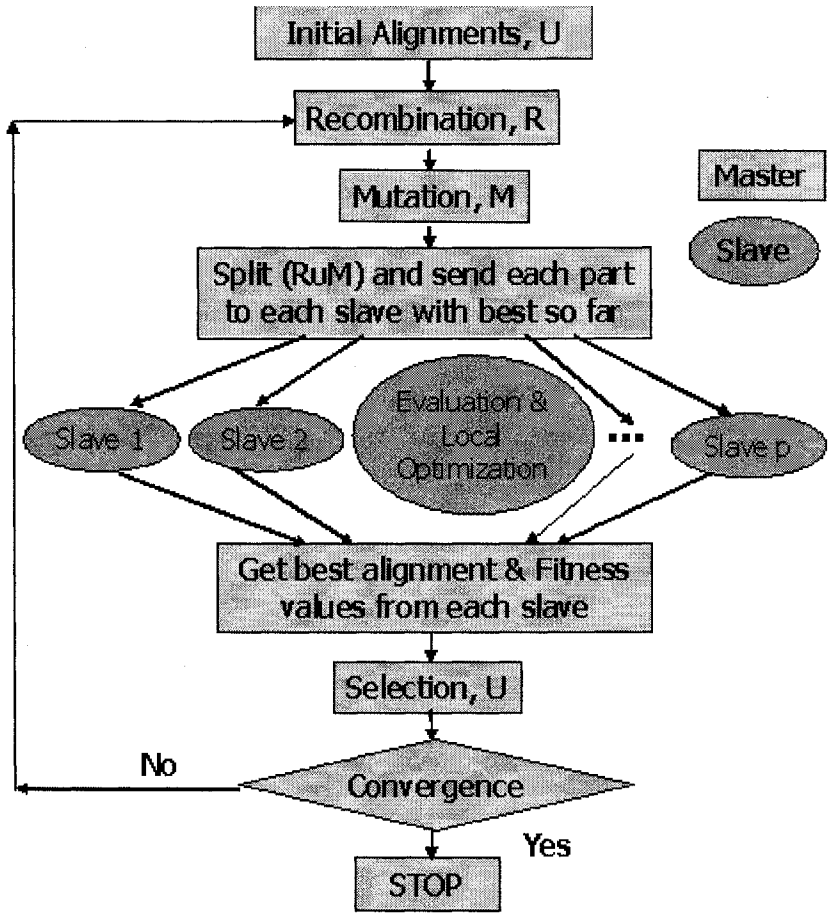

Figure 4.5: Flow chart of SQST-PEST

not improving much for some time) or a maximum number of generation. In SQST-PEST, the evaluation of the $\mu + \lambda$ solutions in the current population is done in parallel. The slaves evaluate $\frac{\mu+\lambda}{p}$ solutions and perform local optimizations independently of each other and in parallel. This approach allows to evolve very large population quickly, as the fitness evaluations and the local optimization are very computationally costly, and thus should be parallelized.

SQST-PEST can only thread one query against one template. We can thread against t templates by calling SQST-PEST t times within a loop, once for each template. This method is called serial SQST-PEST and can be used for threading single query onto multiple template.

1. Master: Given Q and $T_1, T_2, ..., T_t$
   - Split $\{T_1, ..., T_t\}$ into p sets of $T^1, T^2, ..., T^p$
   - Send Q and set T^j to slave S_j, $1 \leq j \leq p$
   - Obtain best solution B_j from each S_j
   - Return $Best\_so\_far \leftarrow$ best in $\{B_1, B_2, ..., B_p\}$
2. Slave S_j: Given Q and $T^j = \{T_1^j, T_2^j..T_{t/p}^j\}$
   - Apply EST on inputs Q and T_i^j, $1 \leq i \leq t/p$
   - Return the best solution so far to the master

Figure 4.6: SQMT-PEST Algorithm

4.2.2 SQMT-PEST Approach

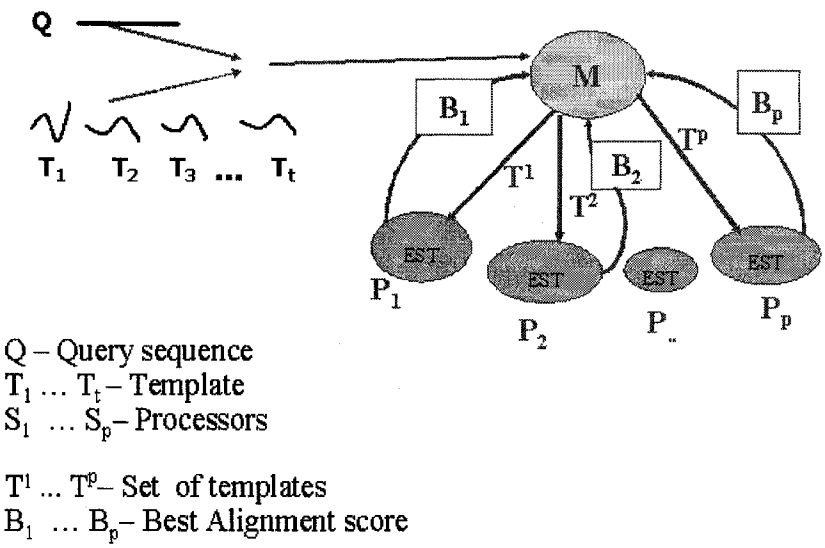

Figure 4.7: Visualization of SQMT-PEST

In order to efficiently thread a query Q against multiple templates $T_1, T_2, ...T_t$, we can call SQST-PEST t times; each time with different template T_i, $1 \leq i \leq t$. We propose another method, the Single Query Multiple Template Parallel ES Threading (SQMT-PEST) for doing fast multiple threading. Figure 4.6 shows the SQMT-PEST algorithm and the parallelization is visualized in Figure 4.7. In the algorithm, there are p slave processors and each slave S_j

receives a distinct subset, T^j of templates from $\{T_1, T_2, ..., T_t\}$ along with the query (each T^j contain t/p templates). A slave S_j will then proceed to apply EST algorithm t/p times, each time with a distinct template from T^j. The master returns the best solution out of the best results obtained from the slaves.

The splitting of the set of templates into different slave processors distributes the load among the slave processors. Since $t \geq p$, each of the first (t mod p) slave processors take $(\frac{t}{p} + 1)$ numbers of templates. The rest of the processors take $\frac{t}{p}$ templates each to make at best-possible even distribution of load. If (t mod p)=0, the load is equally distributed among the processors in respect of number of template. The splitting strategy is shown in Figure 4.8:

- $k = t \bmod p$
- $T^i = \frac{t}{p}$
- for $i = 1..p$
  - If $k > 0$
    - $T^i = \frac{t}{p} + 1$
    - $k = k - 1$
  - Send T^i templates to slave processor, i
- End for

Figure 4.8: Splitting strategy in SQMT-PEST Algorithm

In this approach, each slave runs the EST algorithm $\frac{t}{p}$ times independently of other slaves and in parallel. This allows the possibility of threading a query against large template sets in reasonable time. Also, as a matter of future research, SQMT-PEST will run much faster if the slave use SQST-PEST in place of EST algorithm. Notice also that the slaves do not communicate with each other (and need not to) since they each solve distinct threading problems; the problems are distinct because the templates are distinct proteins with possibly distinct structures.

4.3 Complexity Analysis

The calculation of the fitness value is usually the most costly (in time and space) operation in evolutionary computation, as it involves decoding a solution's representation and processing the solution in its original form in order to obtain its fitness. Given a query Q, a template T and a candidate solution vector S, the asymptotic complexity of the energy $E(S)$ is $O(|T|^2)$; since the computation of the E_{pair} term is the most expensive and that the entry for each pair of positions (or amino acids) in T is searched for in the two dimensional energy matrix of [11]. Both our mutation and recombination algorithms run in $O(|T|)$ time each. In all our experiments, we use $\mu < \lambda$ (where $\lambda = b\mu$, $b > 1$) and $\lambda = a|T|$ ($a > 0$). This helps to define good upper bounds on our algorithm.

The main operations in our algorithms are identified as follows:

Recombination: The recombination needs to create a binary string random number (0, 1) as 'mask'. Depending on the bits in binary string, it chooses the bit from either selected parents to create recombined individual. This operation is linear and it is applied on the length of the template structure, $|T|$, thus, the computation for each individual represents $O(|T|)$. The recombination is applied only on the parents, which is consisted of μ individuals. Thus the complexity of recombination is $\mu O(|T|)$

Mutation: The mutation involves choosing a random offset value and two random position in which the offset value is added to one position while the offset value will be subtracted from other position. Each offspring is generated with a random number of adding and subtracting offsets. The operation is linear and applied to the bits of the individual, thus the computation for each individual represents $O(|T|)$. In each generation, number of total mutants are λ and thus the complexity of mutation operation is $\lambda O(|T|)$.

Fitness evaluation: The fitness evaluation involves pairwise energy computation. It involves to determine the energy from a certain position to all other positions in the place holder of the template structure using two dimensional energy matrix and distance as described in 5.1.1, and get the resultant energy for that certain position. Similarly, we need to

determine the resultant energy for all other positions and total energy score is the sum of all those resultant energies for each position. Since the total position in a solution is $|T|$, the operation $O(|T|^2)$ and since the evaluation is done for $(\mu + \lambda)$ individuals in a generation, total complexity for fitness evaluation is $(\mu + \lambda)O(|T|^2)$

Choosing best so far: Choosing best so far involves comparing the fitness value of $(\mu + \lambda)$ individuals and choose the best. Then it replaces the best so far (if better). Thus the complexity for choosing the best so far operation is $O(\mu + \lambda) + O(|T|)$

Local optimization: Local optimization needs to do mutation of current and best so far, evaluate them and choose the best if improves. Since the operation involves fitness evaluation, so the complexity of this operation is $O(|T|^2)$.

Choosing parents (μ best): Choosing parents involve comparing fitness among $(\mu + \lambda)$ individuals and choose the μ individuals to replace the old parents. Thus the complexity of this operation is $\mu O(\mu + \lambda) + \mu O(|T|)$.

4.3.1 Complexity of EST

In a given generation, the evaluation of solutions and the local optimization contribute the most to the complexity of EST. Summing the complexities of all main operations, we obtain an asymptotic complexity of $(\mu + \lambda)O(|T|^2) + \mu O(\mu + \lambda) = \frac{a+ab}{b}O(|T|^3)$ since $\lambda = b\mu = a|T|$, $(a > 0, b > 1)$.

4.3.2 Complexity of SQST-PEST

In a given operation, we must distinguish between computation time and communication time. Given p slaves, the master sends $\frac{\mu+\lambda}{p} + 1$ solutions and receives the best current solution together with the evaluated fitness of $\frac{\mu+\lambda}{p} + 1$ solutions. Since the master communicates with a slave through a queue under MPI and that it takes O(T) time to send/ receive a solution, therefore, the communication complexity is $\frac{\mu+\lambda}{p}O(|T|) = \frac{a+ab}{bp}O(\frac{|T|^2}{p})$.

The master's computation includes recombination, mutation, and selecting best solution

and next parents. This gives an asymptotic complexity of $(\mu+\lambda)O(|T|)+\mu O(\lambda)+\mu O(p) = \frac{a+ab}{b}O(|T|^2)+\frac{a}{b}O(|T|p)$.

A slave's computation only includes evaluation and local optimization. Therefore, it runs in $\frac{\mu+\lambda}{p}O(|T|^2)$ time, that is $\frac{a+ab}{b}O(\frac{|T|^3}{p})$.

In one generation of SQST-PEST, the local time is the sum of communication time and computation time. SQST-PEST runs in $\frac{\mu+\lambda}{p}O(|T|^2)+\mu O(\lambda+p) = \frac{a+ab}{b}O(\frac{|T|^3}{p})+\frac{a}{b}O(|T|p)$ time. The second term comes from the last statement in the repeat loop of the master. In that statement, we apply "multiple elitist strategy", that is the best solution from all slaves are added to the current population and the best μ current solutions will be used as next parents. Multiple-elitism adds a time overhead as p increases, however, such overhead can be avoided not applying multiple elitism. We can randomly replace a current solution by $best\_so\_far$ and select μ best solutions out of $\mu+\lambda$ current solutions instead of $\mu+\lambda+p$ solutions. Somehow, SQST-PEST can improve EST in search by taking advantage of parallelism with the cost of having an extra time overhead, that is linear with p. If we disallow multiple elitism, then SQST-PEST will be p times faster strategy than EST, given p slaves.

4.3.3 Complexity of SQMT-PEST

Unlike in SQST-PEST, the slaves do all the work in SQMT-PEST. Each slave calls EST sequentially on Q and $\frac{t}{p}$ templates ($t \geq p$) and returns its best solution to the master. Therefore, a slave's computation time is $\frac{t}{p}(\mu+\lambda)O(|T|^2)+\frac{t}{p}\mu O(\mu+\lambda) = \frac{a+ab}{b}O(\frac{|T|^3}{p}t)$.

The master's computation time is $O(t)+O(p)+O(|T|)$; its only task is to send/ receive data to/ from slaves and find the best solution out of p results from slaves. The communication time is $O(\frac{|T|}{p}t)$. The time complexity of SQMT-PEST is therefore $\frac{a+ab}{b}O(\frac{|T|^3}{p}t)+O(p)$. Again, SQMT-PEST is p times faster than EST. Here the master does not repeat, so the overhead of $O(p)$ time due to its last statement occurs only once. Although SQMT-PEST is cubic on $|T|$, it is also linear on t (the number of templates). The serial SQST-PEST (that is SQST-PEST is called sequentially on t templates) has the same complexity as SQMT-PEST.

Chapter 5

Experimental Results and Discussion

In this chapter we will summarize the structure prediction performance of our ES threading method. In literature, several criteria is used to evaluate the performance of structure prediction. Since our research has focused only on alignment algorithm of protein threading, we decided to the following experiments with existing tightest scoring function available. 1. Self threading 2. Comparison with existing methods. Our all computational analysis are based on the score function of Bryant & Lawrence 1993 [11], because it has the highest convergence rate found (99.8%) [37]. The experiments in this chapter illustrate the promise and remaining challenges of protein threading algorithm. In our proposed method of threading, our objective is to determine the optimal alignment, not to test the scoring function.

1. Self threading:

Self threading is a method that determines how good the protein sequence finds it own structure using the protein threading algorithm. In self threading method, the sequence of the template structure is sent as an input of the query sequence for the protein threading to align onto its own structure. It illustrates the effects due to structural environment similarity and propagated pairwise interactions. Self-threading a sequence through its own structural model is an exercise in which alignment errors are explicit and certain behaviors expose very clearly. On the other hand, accurate self-threading across a library of diverse structure types is a challenging task for any current score functions. The determined alignment error to each

secondary structures and comparing optimal energy with its native energy can help to assess the threading model.

2. Comparison with existing methods

The problem representation of our ES protein threading is based on the representation of Yadgari et. al. 2000 [88]. Several interesting problems arise when threading is applied to homologous extension modeling in cases where very little primary sequence similarity remains. These include hydrophobic mismatch, the presence of active site residues in unusual structural environments, and secondary structure length mismatch. The globins are a well studied case among other threading studies, in which a common structure has been conserved while the amino acid sequence has diverged to the point of unrecognizably between some family members [11] [69]. In such situations, we need to check how good protein threading can predict the structural similarity. Yadgai, 2000 published his data for protein threading of sequence and structure taken from homologous protein family. We compared our test results with those data presented by Yadgari 2000. Some data has been changed in Protein Data Bank and our test results are observed on present PDB database.

The strengths of our parallel methods are investigated using different experiments. We found parallel protein threading method in literature review, which is based on linear programming approach [89]. They used a parallel optimization tool, CPLEX software for protein threading. Although [89] used different energy function and approach as us, we have nevertheless compared our approach with [89]'s on their data set. Yanev 2003 reported that the data set, they have used are large enough and never been attempted to test before for the lack of proper algorithm and computational power. We reported our parallel method to attempt for threading on such larger sequences and structures. We will present all details in the following sections.

5.1 Implementation

The protein threading algorithm was implemented in C++ programming language. The parallel implementation was done using Message Passing Interfaces (MPI). The program is interactive for different test conditions and allows for extensive experiments without changing codes. The data set, energy matrices and experiment files are organized in different folders.

5.1.1 Implementation of Fitness Function

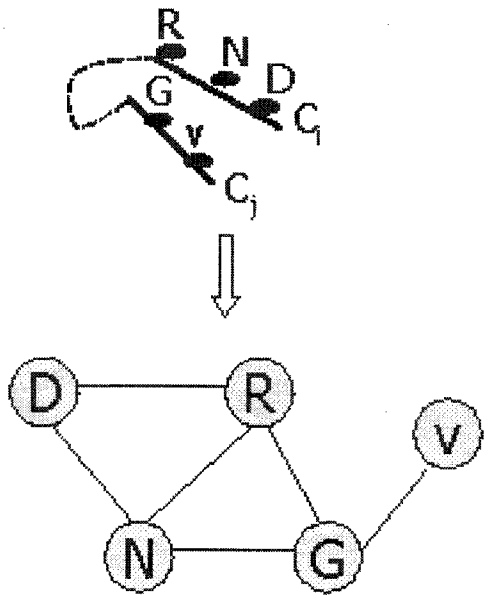

Figure 5.1: Interaction diagram for Pairwise interaction between amino-acids

We have used the distance-specific pair-wise interaction potential values and hydrophobic energies from [11] for scoring the fitness of the alignment of the sequence onto the structure as described in 4.1.2. The method is outlined as: Given an alignment, its total energy (that the energy of its associated fold) is

$E_{total} = E_{single} + E_{pair} + E_{gap}$,

where E_{single} and E_{pair} are computed from the energy matrices of [11], and E_{gap} is the alignment gap penalty function which is set to 3 energy units. E_{total} is a function of the

amino-acid type, the distance between amino-acids, the hydrophobicity of the amino-acids and the alignment gaps.

In a threading alignment, the residues of query sequence occupy the place-holders of the template structure. To determine the energy score for such an alignment, we need to follow the interaction diagram based on the distance between two place-holder position as shown in Figure 5.1. In the figure, C_i and C_j are two segments on a protein sequence and amino acids are shown on it. The nearby amino acids have the interaction or repulsion between them depending on the types of amino acids and the distance in between them. The interaction diagram in Figure 5.1 has shown that the amino acid, D on the C_i has the interaction with R and N. Similarly, R has interaction with D, N and G. On the other hand, G has interaction with R, N and V while V has interaction with only G. The total energy score is the sum of resultant interaction-score of each amino acids onto the template structure, such as D, R, N, G, V. This pairwise interaction depends on the distance between two amino acid. If the distance between two amino acid is more than 10 $\AA$, [11] assumes no interaction between them and the interaction is taken in the intervals of 0-5, >5-6, >6-7, >7-8, >8-9, >9-10$\AA$. The pair-wise potential is shown in Figure 5.2 and 5.3, and the hydrophobic potential which represents the singleton energy of the amino-acid as it occupies a certain position of the template structure is shown in Figure 5.4. All these energy tables are adopted from [11]. The total energy is the sum of pairwise and hydrophobic energy. The two different energy scores can be shown as follows:

Pair-wise Energy, $E_{pair} = \sum_{i=0}^{|T|} \sum_{j=0,|i-j|>5}^{|T|}$Pair-wise Matrix Score ($A_i$, A_j, $dist(i,j)$),

Singleton Energy, $E_{single} = \sum_{i=0}^{|T|} \sum_{j=0,|i-j|>5}^{|T|}$ Hydrophobic Matrix Score (A_i, dist(i,j))

where $dist(i,j)$ = Euclidean distance between α-carbon of position holder i to j in the template. A_i and A_j represents two amino acids occupied at position holder i and j.

The implementation details of energy function is as follows. The interaction potentials are read from the matrices and stored in a three dimensional array in which the first index represents the distance group and other two represents the indices of other interacting amino

	A-A	R-A	R-R	N-A	N-R	N-N	D-A	D-R	D-N	D-D	C-A	C-R	C-N
0-5	0.230	0.237	-0.145	0.158	-0.303	-0.206	0.140	-0.527	-0.534	0.539	0.159	0.174	0.105
5-6	0.350	0.338	0.024	-0.170	-0.258	-0.372	0.068	-0.418	-0.481	0.128	0.008	0.183	0.616
6-7	0.217	0.102	0.218	0.042	0.318	0.211	0.178	-0.609	-0.403	-0.006	-0.031	0.242	0.510
7-8	0.284	0.266	0.249	0.010	-0.331	-0.360	-0.026	-0.243	-0.144	0.036	-0.116	0.001	-0.103
8-9	0.020	0.068	0.185	-0.071	0.115	0.015	0.058	-0.286	-0.121	0.021	0.094	-0.172	0.033
9-10	0.017	0.050	-0.035	-0.150	-0.088	-0.194	-0.121	0.130	-0.075	0.033	0.089	-0.081	-0.044

	C-D	C-C	Q-A	Q-R	Q-N	Q-D	Q-C	Q-Q	E-A	E-R	E-N	E-D	E-C
0-5	-0.117	-0.324	-0.058	0.222	-0.187	-0.128	0.487	-0.430	0.182	-0.713	-0.262	-0.178	0.121
5-6	0.788	0.841	-0.028	-0.297	-0.224	-0.194	-0.158	-0.002	0.194	-0.394	-0.569	0.301	-0.007
6-7	0.509	1.422	0.059	-0.084	-0.296	-0.241	-0.491	0.316	0.155	-0.602	-0.293	-0.123	0.106
7-8	0.456	0.458	-0.030	-0.022	-0.201	-0.143	0.058	0.039	-0.031	-0.112	-0.138	-0.173	-0.117
8-9	0.130	0.507	0.190	-0.273	0.080	-0.043	0.017	-0.250	0.092	-0.083	-0.235	0.248	0.422
9-10	-0.024	1.328	-0.066	-0.222	-0.125	0.001	0.127	0.184	0.117	-0.412	-0.123	-0.157	0.052

	E-Q	E-E	G-A	G-R	G-N	G-D	G-C	G-Q	G-E	G-G	H-A	H-R	H-N
0-5	-0.728	1.060	0.282	-0.037	-0.097	-0.338	-0.187	-0.072	-0.041	-0.068	-0.078	0.111	-0.141
5-6	0.081	-0.218	-0.165	-0.238	0.271	-0.157	0.084	0.088	0.028	-0.188	-0.052	-0.613	-0.319
6-7	0.087	-0.429	-0.099	-0.089	-0.178	-0.166	0.028	-0.177	0.164	-0.210	0.223	0.295	-0.325
7-8	-0.200	0.501	0.059	-0.102	-0.078	-0.167	-0.062	-0.125	0.341	-0.298	-0.044	-0.237	-0.196
8-9	-0.114	-0.086	-0.061	-0.087	-0.170	-0.106	0.105	-0.061	-0.052	-0.122	0.132	-0.238	-0.039
9-10	-0.052	0.062	-0.017	0.017	0.074	-0.047	-0.124	-0.065	-0.089	-0.133	0.163	0.149	0.084

	H-D	H-C	H-Q	H-E	H-G	H-H	I-A	I-R	I-N	I-D	I-C	I-Q	I-E
0-5	-0.278	-0.187	0.247	0.072	0.127	0.392	-0.536	-0.014	0.456	0.330	-0.478	0.030	0.365
5-6	-1.298	-0.431	0.374	-0.196	0.077	0.740	-0.386	-0.111	0.881	0.099	-0.141	0.275	0.537
6-7	-0.189	-0.085	-0.311	-0.616	0.024	0.818	0.108	0.017	0.570	0.275	-0.262	0.039	0.464
7-8	-0.286	0.322	0.056	-0.629	0.247	0.528	-0.051	-0.040	0.398	0.447	-0.298	0.239	0.158
8-9	-0.123	-0.400	-0.054	-0.180	-0.144	0.104	-0.094	0.191	0.081	0.293	-0.091	0.027	-0.019
9-10	-0.087	-0.378	0.067	-0.003	0.072	-0.067	0.032	0.116	0.079	0.148	-0.283	0.078	0.122

	I-G	I-H	I-I	L-A	L-R	L-N	L-D	L-C	L-Q	L-E	L-G	L-H	L-I
0-5	0.127	-0.082	-0.051	-0.249	0.374	0.430	0.336	-0.061	0.391	0.480	-0.006	0.029	-0.233
5-6	0.565	0.080	-0.531	-0.232	0.842	0.325	0.441	-0.072	-0.022	0.288	0.154	0.175	-0.258
6-7	0.260	0.437	-0.698	-0.163	0.113	0.490	0.501	-0.259	0.061	0.308	0.311	0.018	-0.702
7-8	0.114	0.258	-0.298	0.047	0.366	0.360	0.322	-0.317	0.266	0.179	0.115	0.207	-0.347
8-9	0.091	0.233	0.225	0.047	0.050	0.114	0.237	0.031	-0.025	0.084	0.174	0.138	-0.071
9-10	0.181	-0.276	0.092	-0.034	0.052	0.077	0.124	-0.093	-0.056	0.073	0.068	0.252	-0.030

	L-L	K-A	K-R	K-N	K-D	K-C	K-Q	K-E	K-G	K-H	K-I	K-L	K-K
0-5	-0.810	0.063	0.734	0.222	-0.759	1.115	-0.181	-0.782	0.065	-0.236	-0.058	-0.035	0.567
5-6	-0.265	0.429	0.683	-0.124	-0.674	-0.278	-0.173	-0.897	0.239	0.420	-0.075	0.142	0.277
6-7	-0.594	-0.044	0.592	-0.052	-0.615	0.089	0.222	-0.734	-0.325	0.389	0.088	0.278	0.340
7-8	-0.542	-0.016	0.020	-0.044	-0.369	0.021	-0.225	-0.382	-0.198	0.320	0.323	0.342	-0.138
8-9	-0.134	-0.018	0.118	-0.002	-0.263	0.001	0.051	-0.493	0.077	0.188	-0.001	0.072	-0.072
9-10	0.003	0.029	0.211	-0.115	0.044	-0.014	-0.059	-0.118	0.017	-0.039	0.086	-0.017	-0.262

	M-A	M-R	M-N	M-D	M-C	M-Q	M-E	M-G	M-H	M-I	M-L	M-K	M-M
0-5	-0.344	0.509	0.405	0.097	-0.032	0.013	0.188	0.084	0.293	-0.103	-0.253	0.547	-0.006
5-6	0.129	0.324	-0.320	0.044	-0.307	-0.016	0.363	0.126	0.918	-0.208	-0.173	0.188	-0.280
6-7	0.023	-0.323	-0.164	0.525	-0.177	0.543	0.187	0.276	0.452	-0.221	-0.231	-0.089	-0.454
7-8	-0.263	0.267	0.152	0.121	0.163	0.005	0.411	0.062	-0.137	-0.314	-0.178	0.138	0.473
8-9	-0.172	0.058	0.006	0.495	-0.224	0.311	0.155	0.024	-0.016	-0.217	-0.059	0.123	0.392
9-10	-0.087	0.083	0.139	0.035	0.095	0.288	0.230	-0.034	-0.330	-0.287	-0.118	0.105	0.283

	F-A	F-R	F-N	F-D	F-C	F-Q	F-E	F-G	F-H	F-I	F-L	F-K	F-M
0-5	0.132	-0.018	0.338	0.118	-0.093	0.283	0.270	0.497	-0.163	-0.348	-0.460	0.259	-0.801
5-6	-0.378	0.036	0.453	0.703	-0.407	0.352	0.048	0.071	-0.191	-0.313	-0.184	-0.279	-0.087
6-7	-0.244	-0.157	0.362	0.237	-0.439	0.273	0.402	0.357	0.081	-0.326	-0.333	0.029	-0.438
7-8	-0.002	0.014	0.328	0.270	-0.031	0.091	0.089	0.378	-0.334	-0.304	-0.233	0.157	-0.284
8-9	-0.005	0.194	0.330	0.239	-0.055	0.188	0.205	-0.045	-0.198	-0.284	-0.198	0.255	-0.325
9-10	0.087	-0.043	0.031	-0.022	-0.175	0.011	0.186	0.036	0.237	-0.082	-0.201	0.159	-0.153

	F-F	P-A	P-R	P-N	P-D	P-C	P-Q	P-E	P-G	P-H	P-I	P-L	P-K
0-5	-0.183	0.022	-0.176	0.109	0.175	-0.199	-0.545	-0.088	0.127	0.037	0.382	-0.130	0.068
5-6	-0.162	0.124	0.243	-0.050	0.244	0.777	-0.496	0.130	-0.133	-0.242	-0.035	-0.087	0.069
6-7	-0.198	0.080	-0.200	-0.105	0.065	-0.108	0.044	-0.346	-0.233	-0.057	-0.088	0.394	-0.278
7-8	-0.224	-0.090	0.093	-0.062	-0.073	0.222	0.228	-0.316	0.058	-0.265	0.133	-0.058	-0.035
8-9	-0.040	-0.065	-0.589	-0.047	-0.327	0.180	-0.076	-0.225	-0.093	0.444	0.149	-0.058	0.084
9-10	-0.127	0.006	0.052	-0.211	-0.027	-0.043	-0.118	0.008	0.033	-0.167	0.198	0.034	-0.080

(continued)

Figure 5.2: Distance dependant Pair-wise Potential score (adopted from [11])

	P-M	P-F	P-P	S-A	S-R	S-N	S-D	S-C	S-Q	S-E	S-G	S-H	S-I
0-5	-0.104	-0.083	0.093	0.115	-0.263	-0.334	-0.565	-0.164	0.298	-0.207	-0.100	0.080	0.254
5-6	-0.100	-0.018	0.132	0.167	0.121	-0.358	-0.339	-0.188	-0.175	-0.249	-0.076	-0.365	0.431
6-7	0.096	0.458	0.134	-0.118	0.124	-0.588	-0.199	-0.199	-0.126	0.077	-0.255	-0.536	0.476
7-8	-0.252	0.143	0.088	0.056	-0.115	0.065	-0.036	-0.220	0.117	-0.044	-0.166	-0.215	0.153
8-9	-0.167	-0.093	0.281	-0.069	0.011	-0.133	-0.043	-0.015	-0.104	0.003	0.006	-0.071	0.011
9-10	0.023	0.203	-0.124	-0.051	0.105	0.021	-0.030	-0.353	0.083	-0.066	0.012	-0.006	-0.012

	S-L	S-K	S-M	S-F	S-P	S-S	T-A	T-R	T-N	T-D	T-C	T-Q	T-E
0-5	0.390	-0.246	0.259	0.357	0.057	-0.150	0.002	0.250	-0.019	-0.433	0.021	0.060	-0.006
5-6	0.308	-0.099	0.089	0.162	-0.087	0.021	-0.197	-0.187	-0.121	-0.124	0.206	-0.093	0.094
6-7	0.451	0.107	0.062	0.090	-0.022	-0.255	-0.003	-0.040	-0.182	-0.034	-0.010	-0.041	0.061
7-8	0.151	-0.292	-0.011	0.029	-0.117	-0.050	0.005	0.137	-0.003	-0.179	-0.074	0.008	0.089
8-9	0.003	0.049	-0.056	0.341	0.132	-0.027	-0.023	0.085	-0.068	-0.061	-0.244	-0.012	0.050
9-10	0.030	0.057	0.081	0.118	0.010	-0.002	0.084	-0.003	0.124	0.036	0.000	-0.054	-0.056

	T-G	T-H	T-I	T-L	T-K	T-M	T-F	T-P	T-S	T-T	W-A	W-R	W-N
0-5	-0.106	0.061	0.167	0.075	-0.170	0.087	0.058	0.108	-0.430	-0.315	0.117	-0.244	0.536
5-6	-0.306	0.011	0.017	0.205	-0.190	0.296	0.411	-0.151	0.053	-0.359	-0.048	-0.050	0.424
6-7	-0.044	-0.264	-0.073	0.020	0.159	0.483	0.250	-0.038	-0.169	-0.289	-0.213	0.242	-0.120
7-8	-0.252	-0.053	0.097	0.003	-0.059	0.089	-0.061	0.018	-0.056	0.125	-0.191	-0.288	0.062
8-9	-0.117	-0.074	-0.053	0.065	-0.024	0.172	-0.051	0.079	-0.020	0.209	-0.083	0.051	0.141
9-10	-0.165	0.094	0.026	0.007	0.096	-0.209	0.038	0.069	-0.050	-0.065	0.056	-0.221	0.265

	W-D	W-C	W-Q	W-E	W-G	W-H	W-I	W-L	W-K	W-M	W-F	W-P	W-S
0-5	1.287	0.080	-0.459	-0.089	0.036	-0.098	-0.505	-0.635	-0.822	-0.783	-0.185	-0.312	0.296
5-6	0.386	-0.761	0.712	-0.020	-0.138	-0.397	-0.210	-0.008	0.461	-0.351	0.103	-0.161	-0.067
6-7	-0.126	-0.650	0.204	0.153	0.147	-0.700	0.233	-0.283	-0.036	-0.036	-0.240	-0.017	0.396
7-8	0.263	-0.442	-0.244	-0.222	0.277	0.244	-0.105	-0.116	0.218	-0.293	-0.243	0.186	0.231
8-9	-0.148	-0.203	0.112	-0.014	0.144	0.016	-0.037	-0.128	0.009	-0.338	-0.053	0.194	0.025
9-10	0.108	0.113	-0.297	0.179	0.046	0.061	-0.237	-0.009	-0.179	0.093	-0.041	0.020	-0.108

	W-T	W-W	Y-A	Y-R	Y-N	Y-D	Y-C	Y-Q	Y-E	Y-G	Y-H	Y-I	Y-L
0-5	0.040	0.707	-0.030	0.084	-0.491	0.494	0.069	-0.036	-0.004	0.121	-0.305	-0.106	-0.289
5-6	0.383	-0.206	-0.035	0.030	-0.022	0.380	-0.240	-0.092	0.493	-0.492	-0.127	0.291	-0.224
6-7	0.516	0.743	-0.102	-0.186	-0.191	-0.089	0.302	-0.211	0.187	0.047	0.113	0.901	0.085
7-8	0.188	0.715	0.075	0.091	0.126	-0.257	0.253	-0.059	0.100	0.018	0.024	-0.063	-0.087
8-9	-0.032	0.302	-0.030	0.156	0.085	-0.188	-0.215	0.042	0.146	0.184	0.037	-0.057	-0.134
9-10	-0.107	0.473	-0.082	0.072	0.106	-0.197	-0.135	0.132	-0.072	-0.015	0.083	0.039	-0.003

	Y-K	Y-M	Y-F	Y-P	Y-S	Y-T	Y-W	Y-Y	V-A	V-R	V-N	V-D	V-C
0-5	-0.294	-0.218	-0.335	0.071	0.442	0.492	-0.108	0.211	-0.339	-0.011	0.128	0.442	-0.026
5-6	-0.536	-0.296	-0.040	0.041	0.381	0.187	0.036	0.121	0.069	0.459	0.311	0.378	-0.325
6-7	0.011	-0.139	-0.088	0.115	0.139	-0.068	-0.152	0.269	-0.067	0.383	0.385	0.356	-0.143
7-8	0.197	-0.226	0.185	0.032	0.111	-0.128	-0.069	-0.251	-0.037	-0.006	0.179	0.122	0.051
8-9	0.039	-0.069	-0.068	0.082	-0.042	0.144	0.081	0.032	-0.028	0.301	0.234	0.105	0.019
9-10	0.056	-0.172	-0.010	-0.076	-0.013	0.102	0.028	0.229	-0.061	0.047	0.190	0.112	-0.114

	V-Q	V-E	V-G	V-H	V-I	V-L	V-K	V-M	V-F	V-P	V-S	V-T	V-W
0-5	-0.191	0.258	0.141	0.006	-0.253	-0.454	0.253	-0.285	-0.288	0.358	0.316	0.251	-0.036
5-6	0.148	0.159	0.122	0.499	-0.633	-0.592	0.427	-0.358	-0.254	-0.111	0.186	0.039	-0.224
6-7	0.130	0.597	0.098	0.107	-0.453	-0.569	0.035	-0.364	-0.215	0.136	0.403	-0.024	-0.148
7-8	0.026	0.105	0.030	0.189	-0.234	-0.227	0.027	-0.068	-0.116	0.164	0.280	0.056	-0.081
8-9	-0.026	0.036	0.165	0.185	-0.103	-0.043	-0.142	-0.009	-0.168	0.224	-0.077	-0.122	0.012
9-10	-0.005	0.121	0.025	-0.054	-0.068	-0.026	0.015	0.036	-0.078	0.082	0.005	-0.015	-0.093

	V-Y	V-V	p-A	p-R	p-N	p-D	p-C	p-Q	p-E	p-G	p-H	p-I	p-L
0-5	0.075	-0.641	-0.231	-0.133	-0.255	-0.052	-0.365	0.150	0.077	-0.520	0.097	0.512	0.289
5-6	0.122	-0.405	-0.189	-0.014	0.012	-0.031	-0.194	0.005	0.178	0.052	0.065	0.119	0.213
6-7	-0.068	-0.357	-0.124	-0.001	0.089	0.238	-0.218	-0.030	0.189	0.160	0.067	-0.198	0.032
7-8	-0.059	-0.162	0.010	0.090	0.164	0.114	-0.134	0.094	0.201	0.009	-0.069	-0.178	-0.237
8-9	-0.137	-0.341	0.009	0.100	0.075	0.108	0.078	-0.026	0.198	0.105	-0.012	-0.197	-0.174
9-10	-0.074	0.051	0.028	0.005	0.062	0.024	0.021	0.005	0.045	0.102	0.012	-0.088	-0.142

	p-K	p-M	p-F	p-P	p-S	p-T	p-W	p-Y	p-V	p-p
0-5	-0.021	0.391	0.232	0.033	-0.396	-0.154	0.241	-0.045	0.230	0.000
5-6	-0.062	-0.054	0.004	-0.037	0.069	-0.056	-0.030	-0.004	-0.071	0.000
6-7	-0.083	0.090	0.063	0.001	0.006	-0.126	0.049	0.027	-0.252	0.000
7-8	0.090	-0.027	0.073	-0.070	0.126	0.067	-0.088	-0.011	-0.235	0.000
8-9	0.021	-0.072	-0.114	0.020	0.064	0.098	-0.073	-0.033	-0.070	0.000
9-10	0.018	-0.115	-0.105	0.071	0.121	0.053	-0.069	-0.025	-0.034	0.000

[a]Pairwise potential components μ_{ij}^{δ}. Residue types follow standard 1-letter codes, with the peptide group indicated by lower-case "p." Distance intervals are 0–5; >5–6; <6–7; <7–8; <8–9; and <9–10 Å.

Figure 5.3: Distance dependant Pair-wise Potential score (adopted from [11])

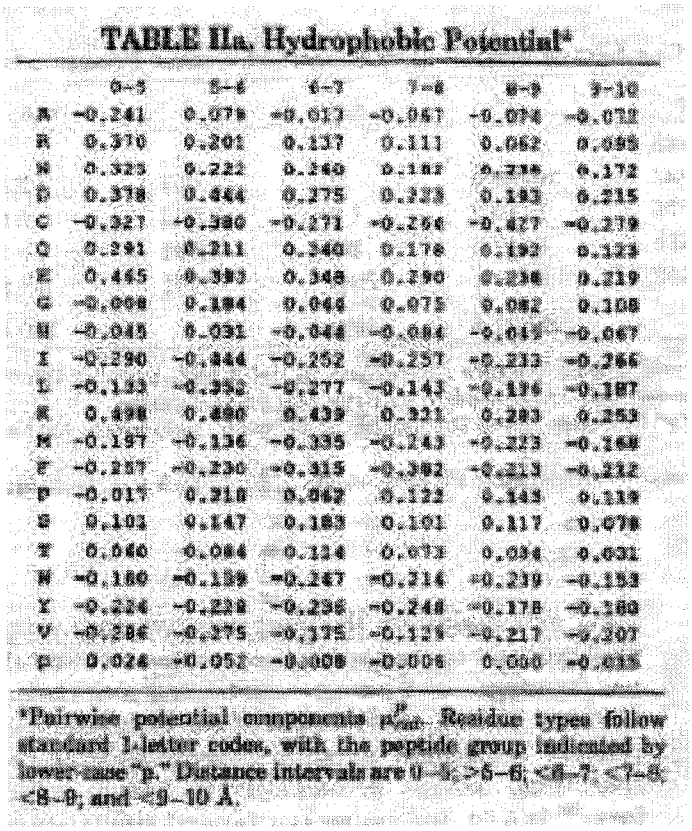

TABLE IIa. Hydrophobic Potential*

	0–5	5–6	6–7	7–8	8–9	9–10
A	−0.241	0.078	−0.013	−0.067	−0.074	−0.072
R	0.370	0.201	0.137	0.111	0.062	0.095
N	0.323	0.222	0.140	0.182	0.218	0.172
D	0.378	0.444	0.275	0.223	0.183	0.215
C	−0.327	−0.380	−0.271	−0.256	−0.427	−0.279
Q	0.291	0.211	0.340	0.178	0.192	0.123
E	0.465	0.383	0.348	0.330	0.236	0.219
G	−0.008	0.184	0.048	0.075	0.062	0.108
H	−0.045	0.031	−0.048	−0.084	−0.045	−0.067
I	−0.290	−0.344	−0.252	−0.257	−0.233	−0.266
L	−0.143	−0.252	−0.277	−0.143	−0.176	−0.187
K	0.498	0.480	0.439	0.321	0.283	0.253
M	−0.197	−0.136	−0.335	−0.143	−0.223	−0.168
F	−0.257	−0.230	−0.315	−0.382	−0.213	−0.232
P	−0.017	0.210	0.067	0.122	0.143	0.119
S	0.101	0.147	0.183	0.101	0.117	0.078
T	0.040	0.084	0.124	0.073	0.094	0.031
W	−0.160	−0.159	−0.267	−0.216	−0.239	−0.153
Y	−0.224	−0.228	−0.236	−0.248	−0.178	−0.180
V	−0.284	−0.275	−0.175	−0.129	−0.217	−0.207
p	0.024	−0.052	−0.008	−0.006	0.000	−0.035

*Pairwise potential components p^{β}_{aa}. Residue types follow standard 1-letter codes, with the peptide group indicated by lower-case "p." Distance intervals are 0–5; >5–6; <6–7; <7–8; <8–9; and <9–10 Å.

Figure 5.4: Singleton Energy score (adopted from [11])

acids. Similarly, the hydrophobic matrices are stored in two dimensional array where the first index represents the distance group and other one is the index of each amino acids. The alignment is represented as a string of integers. Thus the scoring of alignment was done by decoding the string of integers in the sequence of amino-acids and gaps. For each template structure, the atomic coordinates of amino acids are stored in a flat file database. The coordinates of alpha-carbon for each amino-acid is stored in a distance matrix. We used a parser to retrieve the coordinates from the PDB file. The distance between amino acids is the Euclidean distance between the corresponding alpha-carbon atom of amino acids. From each amino acid in the sequence we determine the distance of all other amino acids and group them in the distance intervals. Now based on the distance interval and amino-acid types, we can get the potential value. All other amino acids those are in contact of certain amino-

acid in the alignment are counted to determine the potential energy in the same manner. The resultant of all those contact energies represents the contribution on energy score for a certain amino acid to hold that position. Similarly, summing up all resultant energies for each non-local position gives the energy score of that sequence-structure alignment. We did not consider the energy between neighboring (i.e. local in the sequence) residues. The rational is that neighboring residues will always be in contact (i.e. the physical distance between them is small) and thus they will have the same contribution to the total energy independent on the fold they are in. The nonlocal pairwise interactions omits the contacts between residues whose sequence indices differ by less than 5 as the consideration is taken in [11] [88]. The energy score determined for the alignment is used to calculate the fitness of that alignment as described in 4.1.2.

5.1.2 Implementation of ES

The implementation of evolution strategy starts with implementing the problem representation and it includes to create a set of initial parents. Mutation as discussed in 4.1.3 and recombination as discussed in 4.1.4 are implemented to generate the population for the next generation. The fitness of each individual in the population is determined as described in 5.1.1 and 4.1.2 The parents for the next generation are chosen as the principle of $(\mu + \lambda)$ - ES as described in 2.4.

5.1.3 Implementation of Parallel Architecture

The two architectures for singleton threading and multiple threading is implemented as described in chapter 4. The program is coded in C++ and Message Passing Interfaces (MPI) functions are used to send the jobs in different processors and receive back the results in master-slave architecture as described in 4.2.

5.2 Experimental Environment

The environment of the implementation is used as the SHARCnet (Shared Hierarchical Academic Research Computing Network, http://www.sharcnet.ca/). The SHARCNet is an High Performance Computing (HPC) platform that spans 11 leading academic institutions in South Central Ontario and exists to support leading-edge research. The set up of our programming environment is based on the one of SHARCNet systems, called Tiger located at University of Windsor. The system specification is as follows:

Compaq Alpha ES40 8 $\times$ 833 MHz,

4 GB memory,

Gigabit Ethernet With Fiber Patch network.

The system is ideal for serial/ parallel MPI code development and large computation. But since the SHARCNet did not marge yet in the grid architecture, we are limited to use only 8 processors of Tiger. All our experiments are done in above environment and results are based on that.

5.3 Experimental Data Set

The experimental data set is taken as the real biological data from PDB (Protein Data Bank, http://www.rcsb.org/pdb/), maintained by the Research Collaboratory for Structural Bioinformatics (RCSB), and managed by Rutgers, The State University of New Jersey and the San Diego Supercomputer Center (SDSC), University of California, San Diego (UCSD). This database can be accessed at http://www.rcsb.org/pdb/. The PDB is the biggest database for protein sequence and structural data. Each protein can be accessed in the PDB using the pdb id of that protein. The sequence of amino acids, secondary structures of protein such as α-helix, β-sheet and loops, and atomic coordinates can be accessed in the .pdb formatted file. BioPerl parser or any other self defined parser can be used to extract the coordinate data. The sequence can be downloaded directly in fasta format.

The potential matrices and hydrophobic potential matrices are given in Bryant and Lawrence, 1993 [11].

5.4 Results and Discussion

The most important criterion for a threading algorithm is that it should be good enough to discriminate a query's structure from the other structures from the set of templates. To evaluate the performance of our algorithm, we first applied self-threading with EST and SQST-PEST to see if it returns the correct answer for a given query Q and its own template T (that T is Q with its structure information). In self-threading, the energy of T is called native energy, E_{native}, and can be obtained from the energy matrices of [11] given Q. We compared the native energy of proteins with the optimal energy found by self threading. EST determines a (near-)optimal energy, E_{thr} for given Q and its template T.

Yanev 2003 [89] used a parallel approach for protein threading. Although [89] used different energy function as us, we have nevertheless compared our approach with [89]'s on their data set. Yanev 2003 reported that the data set, they have used, are large enough and have never been attempted before due to slow computation.

Next, we tested the performances of SQST-PEST and SQMT-PEST. We did experiments both on running times and quality of threading, for each parallel algorithm. Furthermore, we compared serial SQST-PEST and SQMT-PEST on quality of threading and times. In serial SQST-PEST, given Q, a set of t templates, we call SQST-PEST t times to thread Q against each template.

5.4.1 System Parameters

The evolution strategy, that we used, falls in the category of $(\mu + \lambda)$-ES. The parameters μ is the number of parents for each generation and λ is the total number of mutants in each generation. μ and λ are set according to the length of a given template, that is $|T|$. We

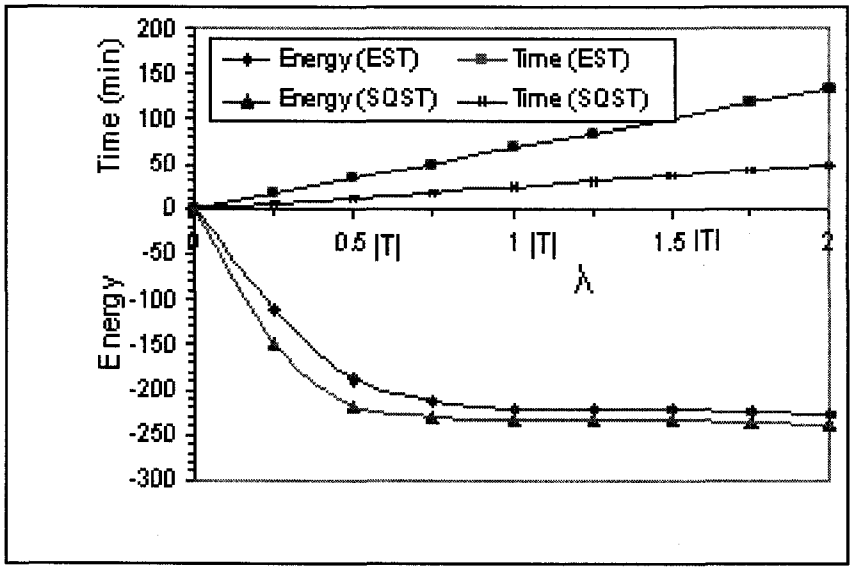

Figure 5.5: EST vs. SQST-PEST as λ increases as a fraction of $|T|$ on threading 1gal(583)-1ad3_a(452)

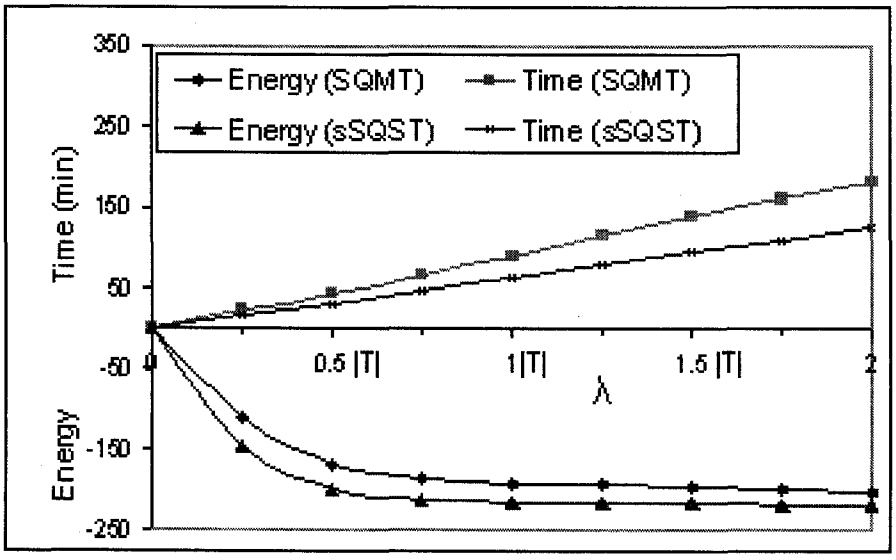

Figure 5.6: SQMT vs. serial SQST-PEST as λ increases as fraction of $|T|$ on query 1bbh_a - templates 451c(82), 1kdu(85), 1tlk(103), 2ccy_a(128), 1eca(136), 1apa(261), 1cca(291)

used μ as $\frac{\lambda}{5}$ and performed the experiment for varying number of mutants as fraction of $|T|$ (i.e., λ as $\frac{|T|}{4}$, $\frac{|T|}{2}$, .., $2|T|$) for 1000 generation. We did the same experiment for all our EST,

SQST-PEST, SQMT-PEST and serial SQST-PEST. EST and SQST-PEST were applied on one of the largest threading 1gal(583)-1ad3_a(452). SQMT-PEST and serial SQST-PEST were experimented on query 1bbh_a - templates 451c(82), 1kdu(85), 1tlk(103), 2ccy_a(128), 1eca(136), 1apa(261), 1cca(291). The threaded energy and computation time is plotted in respect of number of λ (i.e., as a fraction of $|T|$). The experimental result is shown in Figure 5.5 and Figure 5.6. For the parallel methods, we used all 8 processors. All EST, SQST-PEST, SQMT-PEST and serial SQST-PEST show that the computational time increases linearly for increasing number of λ. The threaded quality for each experiment yields with better result (decreases) as the number of λ increases. The result also shows clearly that the better value of threaded energy is obtained at $\lambda \geq |T|$ and the threaded energy does not change significantly as the λ exceeds $|T|$. At the larger λ ($\lambda > |T|$), the computational time increases without improving on the threaded energy. Based on this observation, we consider λ=$|T|$ and based on that we used $(\frac{|T|}{5} + |T|)$-ES for all of our EST, SQST-PEST, SQMT-PEST and serial-SQST. The number of generations in each experiment is 1000. In all our experiments, we run our algorithm 5 times and give the average result of 5 runs as well as the standard deviation. All times are reported in minute, E_{thr} in the table are the energy value.

5.4.2 Experiments with EST

We used EST algorithm to compare its experimental results with existing method. We also performed self-threading to determine the strength of the algorithm. The experiment used $\frac{|T|}{5}$ parents, $|T|$ mutants for 1000 generation. The experimental results are shown as follows:

Threading Comparison With Existing Methods

Table 5.2 and 5.1 compare the results between our EST and the GA method of [88], on the same protein tested in [88]. For a given query and template, we show energy score and the threading time of GA and EST. It should be noted that [88] used 1000 generations and a population size of 300 solutions for their GA. EST outperformed GA on all inputs: it

gave the lowest energy and fastest time. The comparison is shown as $\frac{E_{est}-E_{yadgari}}{E_{yadgari}} \times 100$ and $\frac{T_{yadgari}-T_{est}}{T_{yadgari}} \times 100$. This motivates that the elitist selection of ES algorithm and our algorithm for mutation and recombination are well-adopted for threading problem.

The results of our EST without and with recombination are also reported in the Table 5.1 and 5.2 respectively. The experimental result shows that EST with recombination gives the better quality of threading although it increases in computation time than that of without recombination. The recombination operator combines the genetic materials of two different individuals and transfers the genetic information to the offsprings. The matter of fact that the recombination operator in our algorithm is applied on the parents and only the better individuals in a generation (elitist selection) are chosen as parents for the next generation. Thus it combines the genetic information of two better individuals and it is expected that better parents may recombine to better offsprings. The well adopted recombination performs better in evolutionary computation and the result showed that our recombination fits well for

Table 5.1: EST vs. GA Threading [88] without recombination

Query (Length)	Template (Length)	Yadgari 2000		EST without recomb			
		E_{thr}	Time	E_{thr}	Comp.	Time	Comp
1bbh(131)	2ccy(128)	-175	29	-173.1±4.3	-0.6%	18.3±3.1	36.9%
1ash(147)	1cca(291)	-111	33	-121.2±4.1	9.2%	32.2±3.3	2.4%
2pfl(156)	1kdu(85)	-94	13	-91.2±4.9	-3.0%	9.2 ±2.5	29.2%
2rhe(114)	1tlk(103)	-122	20	-109.5±4.4	-10.2%	11.1±2.5	44.5%
1ccr(112)	451c(82)	-37	14	-39.5±4.6	6.8%	8.1±1.9	42.1%
1rtc(268)	1apa(261)	-491	120	-479.5±5.1	-2.3%	77.5±4.1	35.4%

Table 5.2: EST vs. GA Threading [88] with recombination

Query (Length)	Template (Length)	Yadgari 2000		EST without recomb			
		E_{thr}	Time	E_{thr}	Comp.	Time	Comp
1bbh(131)	2ccy(128)	-175	29	-191.5±3.8	9.1%	22.5±2.9	22.41%
1ash(147)	1cca(291)	-111	33	-165.2±4.1	48.8%	36.1±3.3	-9.4%
2pfl(156)	1kdu(85)	-94	13	-121.4±3.9	29.1%	11.3 ±2.7	1.3%
2rhe(114)	1tlk(103)	-122	20	-141.5±4.1	15.98%	14.3±2.5	28.5%
1ccr(112)	451c(82)	-37	14	-56.1±4.1	51.6%	10.3±2.1	26.4%
1rtc(268)	1apa(261)	-491	120	-513.2±5.1	-4.48%	83.5±4.2	30.4%

this threading problem. We have determined the effect of recombination operator in all our experiments with EST, SQST-PEST and SQMT-PEST.

Self Threading

To evaluate the performance of our algorithm, we applied self-threading with EST to see if it returns the correct answer for a given query Q and its own template T (that T is Q with its structure information). In self-threading, the energy of T is called native energy, E_{native}, and can be obtained from the energy matrices of [11] for given Q. We compared the native energy of proteins with the optimal energy found by self threading. EST determines a (near-)optimal energy, E_{thr} for given Q and its template T.

Table 5.3 shows the results of self-threading with EST. A given row shows the native energy (E_{native}) of a query and the actual lowest energy (E_{thr}) obtained by threading the query against itself, and the accuracy of EST. For a given sequence, the self threading accuracy is calculated as $(1 - |\frac{E_{thr} - E_{native}}{E_{native}}|) \times 100$. EST is 100% accurate when $(E_{thr}) = (E_{native})$. The table shows the self-threading result with EST. The protein 1cca yields less than 100% accuracy, although EST actually gave an energy lower than E_{native}. E_{native} should be $\leq E_{thr}$ but our result shows that $E_{thr} < E_{native}$ for 1cca protein. This is an exceptional behavior of protein that shows that some proteins may not always in global minimum energy in respect of the energy matrices. The experimental result shows that the proteins larger than 400 amino-acids could not find the own template in self threading. Thus, we recommend that our present algorithm for EST should be limited to use in threading for the protein that has length of less than 400. The effect of recombination is found in 1cem(363) where without recombination, the query could not find the template, but it found the template with recombination. All in all, the result show that EST is very accurate and can be trusted for threading proteins against non-self templates.

Table 5.3: Self threading with EST

Sequence (Length)	E_{native}	EST with recomb			EST wout recomb		
		E_{thr}	Result	Time	E_{thr}	Result	Time
1kdu(85)	-14.9	-14.9	100%	10.1	-14.9	100%	8.7
1tlk(122)	-139.5	-139.5	100%	13.6	-139.5	100%	11.8
2ccyA(128)	-79.2	-79.2	100%	21.1	-79.2	100%	18.1
1apa(261)	-122.2	-122.2	100%	78.3	-122.2	100%	73.2
1cca(292)	-119.9	-124.5	96.2%	86.3	-124.5	96.2%	79.5
1cem(363)	-199.2	-199.2	100%	110.2	-193.5	97.1%	100.1
1gpl(432)	-211.3	-211.3	100%	129.5	-203.8	96.4%	116.6
3grs(478)	-176.1	-173.4	98.5%	139.1	-167.1	94.8%	123.5

5.4.3 Experiments with SQST-PEST

The SQST-PEST aims to thread on the larger protein sequences and templates with better time and threading quality. The strength of SQST-PEST has been tested using self threading with the larger proteins found in PDB. We also did the experiments with SQST-PEST for large threading examples reported in [89]. The performance of SQST-PEST on different number of processors are also reported.

Self-threading with SQST-PEST

The SQST-PEST has been applied on the larger protein where our EST failed to find the own template in self-threading examples. We have used 8 processors for all these experiments and the result is shown in Table 5.4. The experimental result shows that SQST-PEST successfully finds the native structure in self threading where some proteins (larger than 400 sequence-length) failed in EST. As it can be noticed that our algorithm for SQST-PEST described in 5.7 uses the local optimization for the current best result in each slave processors. Then the current best from each slave processor is sent to the master processor where all the current bests are compared and finally updates the global best. Thus the parallelization of expensive fitness function and local optimization in each slave processor empowered SQST-PEST to successfully find the own template for the larger protein as large as of 700 sequence-length in

an affordable time. Our other experiments indicate that using of more slave processors for larger protein may exceed this limit of size of protein and will yield with better threading quality as well as computation time.

Table 5.4: Self threading with SQST-PEST

Sequence	E_{native}	EST with recomb			EST wout recomb		
(Length)		E_{thr}	Result	Time	E_{thr}	Result	Time
1cem(363)	-199.2	-199.2	100%	31.4	-199.2	100%	27.8
1gpl(432)	-211.3	-211.3	100%	38.5	-211.3	100%	33.6
3grs(478)	-176.1	-176.1	100%	41.1	-176.1	100%	37.5
1gal(583)	-231.1	-231.1	100%	52.6	-231.1	100%	47.1
1w63_A(618)	-273.2	-273.2	100%	58.1	-273.2	100%	51.6
101g_A(722)	-366.8	-366.8	100%	68.3	-359.6	98.1%	61.1
1mvw_A(840)	-311.9	-298.7	95.7%	81.2	-291.6	93.49%	69.7

Results of SQST-PEST for Large Proteins

Table 5.5: SQST-PEST vs. Yanev [89] without recombination

Query	Template	# of	Yanev	SQST wout recomb		
(Length)	(Length)	proc.	Time	E_{thr}	Time	Comparison
2bmh(455)	1cem(363)	4	11.6	-298.1±5.1	18.3±2.1	-57.8%
3min(522)	1gpl(432)	4	13.3	-392.2±4.7	22.2±3.1	-66.9%
2cyp(294)	3grs(478)	5	34.2	-251.3±4.2	21.1 ±2.5	38.3%
1gal(583)	1ad3_a(452)	5	43.2	-212.5±4.5	23.2±3.1	46.3%

Table 5.6: SQST-PEST vs. Yanev [89] with recombination

Query	Template	# of	Yanev	SQST with recomb		
(Length)	(Length)	proc.	Time	E_{thr}	Time	Comparison
2bmh(455)	1cem(363)	4	11.6	-323.5±3.3	23.2±1.8	-100%
3min(522)	1gpl(432)	4	13.3	-419.1±3.8	29.5±2.1	-121.8%
2cyp(294)	3grs(478)	5	34.2	-271.3±3.5	26.3±3.1	23.1%
1gal(583)	1ad3_a(452)	5	43.2	-228.2±3.4	31.4±2.9	27.3%

We compare SQST-PEST with the approach discussed in [89]. Yanev [89] implemented a parallel linear programming approach for protein threading on some large proteins. However,

Yanev used different energy function and different approach than SQST-PEST; so we can only compare threading times to see how affordable is our computation time. Table 5.5 and 5.6 shows the performance of SQST-PEST with and without recombination on the proteins used in [89]. The comparison is made as $\frac{T_{yanev}-T_{sqst}}{T_{yanev}} \times 100$ SQST-PEST gave results comparable to Yanev's; sometimes they are better though some others are worse. Notice also that we achieved better times than Yanev on its two hardest pairs of query-template's (see last two rows).

Performance of SQST-PEST

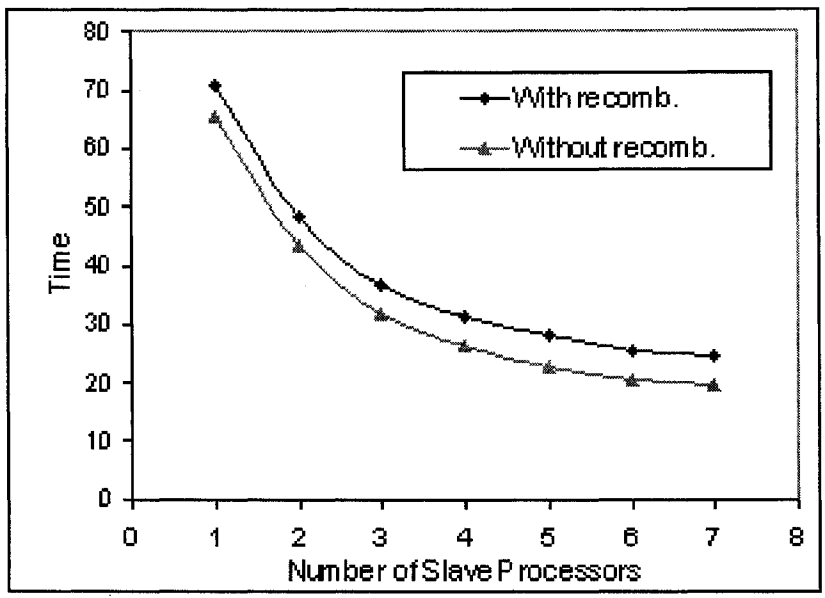

Figure 5.7: Performance of SQST-PEST on 1gal(583)-1ad3_a(452) as p increases.

Next we reported the performance of SQST-PEST on the hardest pair of query and template used in [89] (see the last row of table 5.6), as the number of slave processors increases from 1 to 7. The pair is (1gal, 1ad3_a) and was the hardest threading task in [89]. Figure 5.7 shows the performance in time of threading the query 1gal against the template 1ad3_a with and without recombination as the number of slave increases. This is consistent with the time complexity of SQST-PEST. Table 5.7 shows the performance in energy for the same input

Table 5.7: Performance of SQST-PEST on 1gal(583)-1ad3_a(452) as p increases.

# of proc.	SQST with recomb		SQST without recomb	
	E_{thr}	Time	E_{thr}	Time
2	-216.8±3.8	70.5±1.9	-196.1±4.4	64.4±2.1
3	-220.1±4.1	48.5±2.1	-202.4±3.4	42.1±1.8
4	-224.2±3.9	36.4±2.9	-206.1±4.6	30.6±2.6
5	-228.2±3.4	31.4±2.7	-211.6±3.4	25.3±2.3
6	-232.6±4.1	28.5±1.9	-215.5±4.1	23.8±1.8
7	-233.3±3.3	25.4±1.8	-219.9±3.8	20.6±2.1
8	-234.8±3.4	23.6±1.6	-221.5±4.3	18.1±1.9

pair as we add more slaves. As one can see adding more slaves yields better energy. This is due to the improved search strategy of SQST-PEST given more slaves. We also reported the results of SQST-PEST with and without recombination in Figure 5.7 and in Table 5.7. The effect of recombination improved the quality in SQST-PEST but did not have effect on time and energy in increasing the number of slaves. This also complies with our SQST-PEST algorithm since the algorithm does the recombination operation in the master processor.

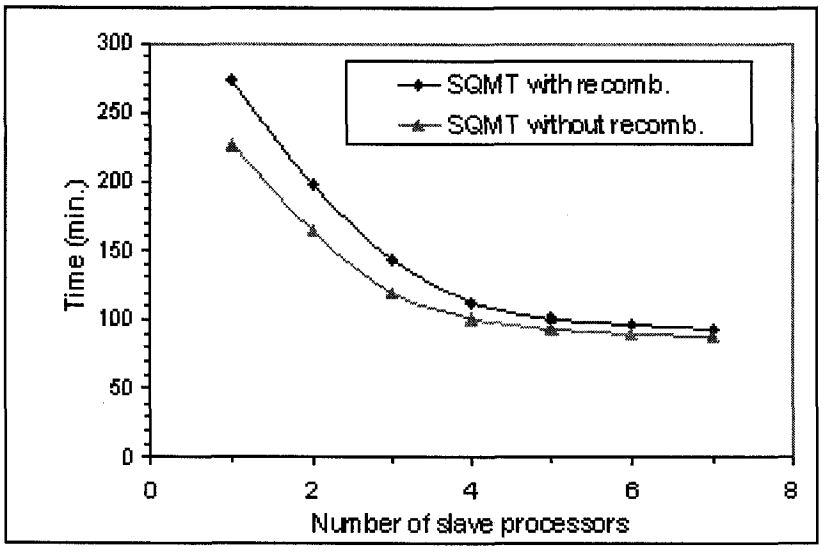

Figure 5.8: Performance of SQMT-PEST (with and without recombination) on query 1bbh_a - templates 451c(82), 1kdu(85), 1tlk(103), 2ccy_a(128), 1eca(136), 1apa(261), 1cca(291) as p increases.

5.4.4 Experiments with SQMT-PEST

The SQMT-PEST aims to thread single query with Multiple templates. We tested SQMT-PEST on query 1bbh_a with 7 different templates 451c(82), 1kdu(85), 1tlk(103), 2ccy_a(128), 1eca(136), 1apa(261), 1cca(291) with different number of slave processors. The performance in time of SQMT-PEST with and without recombination is shown in Figure 5.8 in terms of computing time with varying number of slave processors. The execution time of SQMT-PEST depends on the size of the largest template. Recall that each slave calls EST many times and sequentially on number of input templates. Thus some slaves will work harder (run longer) than other slaves, and the slave that receives the largest template or set of larger proteins will run the longest. SQMT-PEST can be well benefited if each slave processes the same number of templates, that if $t \ mod \ p = 0$ and well distributed with the length of proteins.

Table 5.8: Performance of SQMT-PEST on query 1bbh_a - templates 451c(82), 1kdu(85), 1tlk(103), 2ccy_a(128), 1eca(136), 1apa(261), 1cca(291) as p increases.

# of proc.	SQMT with recomb		SQMT without recomb.	
	Best-fit Template: E_{thr}	Time	Best-fit Template: E_{thr}	Time
2	2ccy_a: -191.5±3.5	273.2±2.8	2ccy_a: -176.1±4.1	233.1±2.2
3	2ccy_a: -192.1±3.7	197.8±3.1	2ccy_a: -176.8±3.9	163.7±2.4
4	2ccy_a: -192.4±4.1	144.7±2.6	2ccy_a: -177.1±3.9	118.4±2.6
5	2ccy_a: -191.5±4.5	112.7±2.9	2ccy_a: -177.5±4.5	100.5±1.9
6	2ccy_a: -192.8±4.1	100.2±2.1	2ccy_a: -177.2±4.1	94.3±2.1
7	2ccy_a: -193.2±4.1	96.5±2.9	2ccy_a: -178.2±4.2	90.2±2.7
8	2ccy_a: -193.5±3.6	91.7±3.5	2ccy_a: -178.5±3.6	87.3±2.9

Table 5.8 shows the energy results of SQMT-PEST on the same data set, and for each value of p. It returned the same best-fit template without significant improving of energy values. It also complies with the algorithm since, in SQMT-PEST, each slave performs the same EST for the templates sent by the master but slaves are working independently. The experimental result shows that the effect of recombination in SQMT-PEST gives better quality in threading but does not affect on varying number of processors. But the computation time with recombination takes longer compare to without recombination if less number of slaves

are used. This is because the recombination is done in slave processors in SQMT-PEST.

5.4.5 Comparisons between SQMT-PEST and Serial SQST-PEST

Table 5.9: Serial SQST-PEST vs. SQMT-PEST with 7 slaves

Query (Length)	Template (Length)	SQMT-PEST		Serial SQST-PEST			
		E_{thr}	Time	E_{thr}	Comp.	Time	Comp.
1bbh_a (131)	2ccy_a(128)	-193.5±4.1	91.7 ±3.5	-215.3±3.6	11.3%	63.9 ±3.1	30.3%
	1apa(261)	-183.2±3.2		-194.2±3.1	5.8%		
	1cca(291)	-164.4±3.8		-172.1±3.6	4.7%		
	1kdu(85)	-134.1±3.4		-145.3±3.8	8.4%		
	1tlk(103)	-131.2±3.3		-136.5±3.3	4.1%		
	451c(82)	-129.1±3.2		-137.2±4.1	6.3%		
	1eca(136)	-116.6±3.1		122.1±4.1	4.7%		

SQST-PEST can only thread one query against one template. We can thread against t templates by calling SQST-PEST t times within a loop, once for each template. We compared the performances of serial SQST-PEST and SQMT-PEST on the same data set as in the previous section. We returned the threading result on query 1bbh_a with 7 different templates 451c(82), 1kdu(85), 1tlk(103), 2ccy_a(128), 1eca(136), 1apa(261), 1cca(291) for both methods. Table 5.9 shows the results of both algorithms with 7 slave processors for threading protein 1bbh_a against 7 different templates. The threading quality in serial SQST-PEST is better in all 7 threading compare to SQMT-PEST. Again serial SQST-PEST is faster than SQMT-PEST. The comparison of threaded energy and time is shown as $\frac{E_{ssqst}-E_{sqmt}}{E_{sqmt}} \times 100$ and $\frac{T_{sqmt}-T_{ssqst}}{T_{sqmt}} \times 100$ respectively. In serial SQST-PEST, since we are using SQST-PEST to thread onto each of the template structures, each threading is done on 7 different slave processors. This facilitates serial SQST-PEST to use the power of better local optimization in each of 7 slave processors (as SQST-PEST algorithm) and that's the reason it outperforms SQMT-PEST which allows to perform EST in each slave processor only.

In the set of template structures all templates are of different length and this causes unbalance in SQMT-PEST since we do not split the structure information in different slave

processors (i. e., we send one structure to a slave). In SQMT-PEST, the slave with the largest protein will work longer than other slave processors. On the other hand, in serial SQST-PEST, it performs the expensive fitness evaluation simultaneously in different slave processors. The splitting of offsprings for each generation is better balanced than SQMT-PEST. It contributes on running SQMT-PEST slower than serial SQST-PEST.

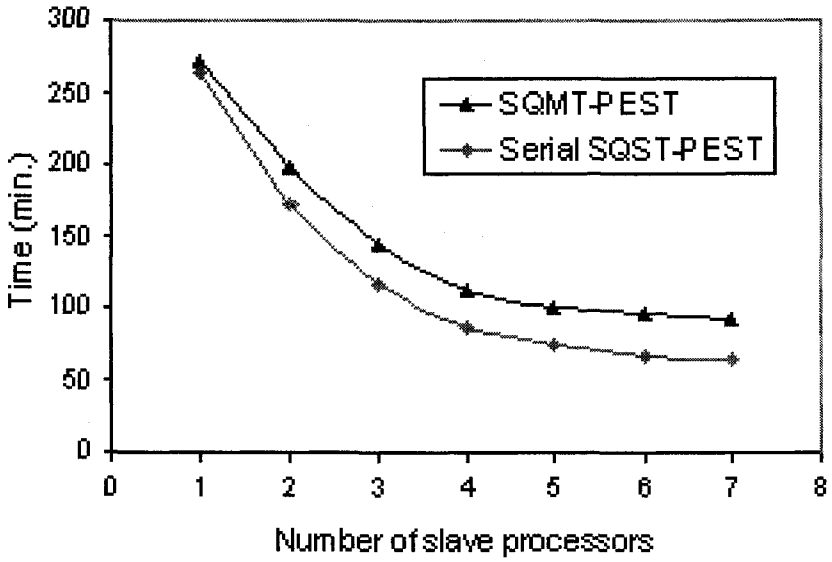

Figure 5.9: Serial SQST-PEST vs. SQMT-PEST as p increases

Table 5.10: Serial SQST-PEST vs. SQMT-PEST as p increases

# of processors	Best-fit Template: E_{thr}		
	SQMT-PEST	Serial SQST-PEST	Comparison
2	2ccy_a: -191.5±3.5	2ccy_a: -191.6±3.8	.05%
3	2ccy_a: -192.1±3.7	2ccy_a: -198.1±4.1	3.12%
4	2ccy_a: -192.4±4.1	2ccy_a: -204.3±3.3	6.2%
5	2ccy_a: -191.5±4.5	2ccy_a: -210.1±3.1	9.7%
6	2ccy_a: -192.8±4.1	2ccy_a: -212.6±4.1	10.3%
7	2ccy_a: -193.2±4.1	2ccy_a: -214.1±3.6	10.8%
8	2ccy_a: -193.5±3.6	2ccy_a: -215.3±3.6	11.2%

Figure 5.9 shows the performance in time of both algorithms as the number of slave increases from 1 to 7. Although serial SQST-PEST is faster than SQMT-PEST for all values

of p, both algorithms suffer the load imbalance problem. In serial SQST-PEST, the unbalance happen when p does not divide $(\mu + \lambda)$ and thus some slaves work larger than other slaves; therefore, for two consecutive values of p, the reduction in running time is small. we can take a better advantage of SQST-PEST if $(\mu + \lambda)\ mod\ p = 0$ and $t\ mod\ p = 0$. In SQMT-PEST, the different sizes of the template structures is also a concern. The slave with the largest protein or with a set larger proteins will cause unbalance in SQMT-PEST and work longer than other slave processors. That's one of the reason that serial SQST-PEST performs faster for all values of slave processors, p.

Table 5.10 show the predicted structure for each value of p on the same data set for both algorithm; they both returned the same structure prediction with serial SQST-PEST giving better energy values. It also clearly noticeable that as p increases, the threading quality is increasing with serial SQST-PEST while SQMT-PEST did not have significant change in energy value. This is the same reason as SQST-PEST performs better quality than EST.

Chapter 6

Conclusion and Future Directions

6.1 Summary of Work Done

The evolutionary strategy, applied on protein threading involves several aspects: problem representation, designing efficient genetic operators such as mutation and recombination, using the fitness criteria based on the energy function that evaluates the individuals for surviving in the next generation. We proposed an evolutionary strategy of $(\mu + \lambda)$-ES and implemented the ES threading method. Our system parameters in protein threading and ES with our proposed recombination performed better in the quality of result and in computational time than similar approaches involved in literature review. The proposed ES with recombination performed better in threaded energy though its execution is slower than without recombination. The experiments with self-threading show the strength of our algorithm both for EST and SQST-PEST.

Since protein threading involves large sequential computation and repeating the same operation with a large number of different data sets without further communication, it is well suited for parallel computation. We proposed and implemented two different methods of parallelization and both of them contributed in reduction of large sequential time for protein threading. Our method, SQMT-PEST is adopted for involving protein threading in structure prediction, where it needs to thread the query sequence with multiple structures in

the template database. On the other hand, SQST-PEST facilitates to attempt in threading large sequences, which were rarely been attempted before due to lack of a suitable algorithm and lack of computational power.

The experiments illustrate that both SQST-PEST and SQMT-PEST reduce large computation time. To facilitate the SQST-PEST method in structure prediction, we used SQST-PEST to run in serial with multiple structures. The method performed better than the SQMT-PEST method that splits different templates in different processors.

Though what we have shown are some crude experiments, they are sufficient to indicate that our method of protein threading can be well suited for structure prediction.

6.2 Limitation and Future Work

The results that we have presented, indicate that evolutionary strategy is a promising approach to the protein threading problem. Although the threading shown here may not be guaranteed to find optimal and consistently good results since the problem is very complicated and search space is huge. But the algorithm consistently finds the better score than structural alignment and the genetic algorithm based method proposed in [88]. It also consistently finds the structure in self threading for certain lengths of templates. Since we have used evolutionary strategy instead of genetic algorithm, the strategy for overcoming local optima is a concern. Our strategy of using local method explained in 4.1.5 performed better when attacking local optima.

Choosing mutation offset to be part of the length of sequence (Q/4) gives the diversity of descendants and hunts for possible solutions. We plan to continue investigating on the mutation offset, other mutation operations and use of some other suitable recombination operation for generating descendants more efficiently.

The parallelization technique we have developed, is suitable for attacking large sequential computations of database search in the protein threading problem. We are aware of the fact that threading a large sequence with large structure is still a computational challenge both

for correctness and time. There are some large data sets of protein sequence which are never been attempted. The evolutionary strategy, by nature is well suited to parallel computation of criteria evaluation and our SQST-PEST performed better in that respect. The multiple local optimization in SQST-PEST showed significant improvement in searching for the best alignment. Thus, we can test our SQST-PEST program in greater number of processors for larger queries and template structures.

Our future plan is to combine SQMT-PEST and SQST-PEST with an efficient parallel program so that while the program simultaneously does the threading with multiple structures, the threading of query with a structure will also be done simultaneously in parallel processors (i. e. each slave of SQMT-PEST will run a SQST-PEST instead of EST). This may contribute to a larger reduction of time to overall protein-threading-based structure prediction.

Load balancing in the slave processors is a concern for the performance of parallel computing. The fact of serial SQST-PEST and SQMT-PEST is that they are having with similar complexity but serial SQST-PEST runs faster. One of the reason behind that is because of load balancing. This also opens up the issue of future research. A template database of equal-size of structures can be used to see the performance of SQMT-PEST and serial SQST-PEST. Load balancing in parallel methods can be improved by sending work on demand to idle processors.

The ES threading and its parallel techniques can be extended further for future research in protein folding and some other protein structure prediction methods. Heuristics may be used to generate initial parents for efficient searching in protein threading. By extending this threading concept with partially determined X-ray/ NMR data may improve structure prediction. Solution obtained from ES threading could be a good starting point of ab-initio method. We can improve the parallelization methods using some hybrid approaches.

Bibliography

[1] T. Akutsu. Protein structure alignment using dynamic programming and iterative improvement. *IEICE Trans. on Information and Systems*, E79-D:1629–1636, 1996.

[2] T. Akutsu and S. Miyano. On the approximation of protein threading. *Theoretical Computer Science*, 210(2):261–275, 1997.

[3] T. Akutsu and K. L. Sim. Protein threading based on multiple protein structure alignment. *Genome Informatics*, 10:23–29, December 1999.

[4] S. F. Altschu, T. L. 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 Acid Res 1997; 25:3389?402*, 25:3389–402, 1997.

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

[6] H. M. BerMan, J. Westbrook, Z. Feng, G. gilliland, T. N. Bhatt, H. Weissig, I.. N. Shindyalov, and P. E. Bourne. The protiein databank. *Nucleic Acids Research*, 28(1):235–242, 2000.

[7] J. R. Biekowska, R. G. Rogers Jr., and T. F. Smith. A method for optimal design of a threading scoring function. In *Proceedings of the third annual international conference on Computational molecular biology*, pages 25–32, 1999.

[8] C. L. Brooks, M. Karplus, and B. M. Pettitt. Proteins: Atheoretical perspective of dynamics, structure and thermodynamics. *John Willey and Sons, New York*, 1990.

[9] Dr. S. H. Bryant. Evaluation of threading specificity and accuracy. *Proteins: Structure, Function, and Genetics*, 26(2):172–186, October 1996.

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

[11] 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.

[12] S. K. Burley, S. C. Almo, J. B. Bonanno, M. Capel, M. R. Chance, T. Gaasterland, D. Lin, A. Sali, F. W. Studier, and S. Swaminathan. Structural genomics: Beyond the human genome project. *Natural Genetics*, 23:151–157, 1999.

[13] T. L. Chiu. *Optimizing potentials for protein structure prediction, inverse protein folding and protein folding.* PhD thesis, UNIVERSITY OF MICHIGAN, 1999.

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

[15] C. Clementi, H. Nymeyer, and J. N. Onuchic. Topological and energetic factors: what determines the structural details of the transition state ensemble and 'on-route' intermediates for protein folding? an investigation of small globular ptoteins. *Jopurnal of Molecular Biology*, 298:937–953, 2000.

[16] J. Digalakis and K. Margaritis. Parallel evolutionary algorithms on mpi. citeseer.ist.psu.edu/585825.html, 2003.

[17] R. L. J. Dunbrack, D. L. Gerloff, M. Bower, X. Chen, O. Lichtrage, and F. E. Cohen. Meeting review: The second meeting on the critical assessment of techniques for protein structure prediction (casp2). *Fold Des*, 2(27-42), 1997.

[18] D. Fischer. 3d-shotgun: A novel, co-operative, fold recognition meta predictor. *Proteins: Structure, Function and Genetics*, 51:434–441, 2003.

[19] D. Fischer, A. Elofsson, L. Rychlewski, F. Pazos, A. Valencia, B. Rost, A. R. Ortiz, and R. L. Dunbrack Jr. Cafasp2: The second critical assessment of fully automated structure prediction methods. *PROTEINS: Structure, Function, and Genetics Suppl*, 5:171–183, 2001.

[20] D. Fischer, L. Rychlewski, R. L. Dunbrack Jr., A. R. Ortiz, and A. Elofsson. Cafasp3: The third critical assessment of fully automated structure prediction methods. *PROTEINS: Structure, Function, and Genetics*, 53:503–516, 2003.

[21] L. J. Fogel, A. J. Owens, and M. Walsh. Artificial inteligence through simulated evolution. *New York: John Wiley*, 1996.

[22] A. Godzik, A. Kolinski, and J. Solnick. A topology fingerprint approach to inverse protein folding problem. *JMB*, 19:227–238, 1992.

[23] W. Gropp, H.S. Lederman, A. Lumsdaine, E. Lusk, B. Nitzberg, W. Saphir, and M. Snir. *MPI - The Complete Reference, Volume 2, The MPI Extensions.* Te MIT Press, 1998.

[24] J. H. Holland. Outline for a logical theory of adaptive systems. *J Assoc. Comput. Mach*, 3:297–314, 1962.

[25] E. S. Huang and S. Subbiah. Using a hydrophobic contact potential to evaluate native and near-native folds generated by molecular dynamics simulations. *J Mol Biol*, 257(3):716–25, 1996.

[26] IBM. *IBM Optimization Solutios and Library Guid and Reference.* IBM corporation, 3 edition, 2001.

[27] ILOG. *CPLEX 7.1 Reference Manual.* ILOG, Worldwide Information Centre, Montain View, CA, March 2001.

[28] Sippl M. J. Knowledge based potentials for proteins. *Current Opinion in Structural Biology*, 5(2):229–235, 1995.

[29] D. Eisenberg J. U. Bowie, R. Luthy. A method to identify protein structures that fold into a known three dimensional structure. *Science, New Series*, 253(5016):164–170, Jul 1991.

[30] D. T. Jones, M. Tress, K. Bryson, and C. Hadley. Successful recognition of protein folds using threading methods biased by sequence similarity and predicted secondary structure. *Proteins: Structure, Function, and Genetics*, 37(S3):112–120, 1999.

[31] L. N. Kinch, J. O. Wrabl, S. S. Krishna, I. Majumdar, R. I. Sadreyev, Y. Qi, J. Pei, H. Cheng, and N. V. Grishin. Casp5 assessment of fold recognition target predictions. *Proteins: Structure, Function, and Genetics Fifth Meeting on the Critical Assessment of Techniques for Protein Structure Prediction*, 53(S6):385–409, October 2003.

[32] J. R. Koza. Genetic programming: On the programming of computers by means of natural selection. *MIT Press, Cambrigse, MA*, 1992.

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

[34] R. Lathrop, J. Bienkowska, B. Bryant, L. Butorovic, C. Gaitatzes, R. Nambudripad, J. White, and T. Smith. *Analysis and Algorithms for Protein Sequence-Structure Alignment*, volume chapter 12, chapter 1998, pages 227–283. Computational Methods in Molecular Biology Elsevier Press, Amsterdam, 1998.

[35] R. H. Lathrop. The protein threading problem with sequence amino acid interaction preferences is np-complete. *Protein Eng.*, 7(9):1059–68, Sep 1995.

[36] R. H. Lathrop and T. F. Smith. A branch-and-bound algorithm for optimal protein threading with pairwise (contact potential) amino acid interactions. *Biotechnology Computing, Proceedings of the Twenty-Seventh Hawaii International Conference*, 5:365–374, January 1994.

[37] R. H. Lathrop and T. F. Smith. Global optimum protein threading with gapped alignment and empirical pair score functions. *J Mol Biol*, 255(4):641–65, Feb 1996.

[38] A. M. Lesk and D. R. Bosell. Homology modelling: inferences grom tables of aligned sequences. *Current opinion in Structureal Biology*, 2:242–247, 1992.

[39] J. T. Linderoth. *Topics in Parallel Integer Programming.* PhD thesis, Georgia Institute of Technology, August 1998.

[40] L. 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.

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

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

[43] V. N. Maiorov and G. M. Crippen. Learning about protein folding via potential functions. *Proteins*, 20(2):167–73, 1994.

[44] V. Mairov and G. Crippen. Contact potential that recognizes the correct folding of globular ptotein. *J. Mol. Biol.*, 227:876–888, 1992.

[45] A. Marin, J. Pothier, K. Zimmermann, and J. F. Gibrat. Frost: A filter based recognition method. *PROTEINS*, 49(4):493–509, 2002.

[46] L. A. Mirny and E. I. Shakhnovich. Residue-residue potentials with a favorable contact pair term and an unfavorable high packing density term, for simulation and threading. *J Mol Biol*, 256(3):623–44, 1996.

[47] L. A. Mirny and E. I. Shakhnovich. Protein structure prediction by threading. why it works and why it does not. *J Mol Biol.*, 283(2):507–26, Oct 1998.

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

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

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

[51] J. P. Overington. Comparison of three dimensional structures of homologous proteins. *Current opinion in Structureal Biology*, 2:394–401, 1992.

[52] T. N. Petersen, C. Lundegaard, M. Nielsen, H. Bohr, J. Bohr, S. Brunak, G. P. Gippert, and O. Lund. Prediction of protein secondary structure at 80% accuracy. *Proteins*, 41:17–20, 2000.

[53] A. A. Rabow and H. A. Scheraga. Improved genetic algorithm for the protein folding problem by use of a cartesian combination operator. *Protein Science*, 5:1800–1815, 1996.

[54] I. Rechenberg. Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. *Stuttgart: Frommann-Holzboog Verlag*, 1973.

[55] C. A. Rohl, C. Strauss, K. Misura, and D. Baker. Protein structure prediction using rosetta. *Methods in Enzymology*, 383:66–93, 2004.

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

[57] B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *J. Mol. Biol.*, 232:584–99, 1993.

[58] B. Rost, R. Schneider, and C. Sander. Protein fold recognition by prediction-based threading. *JMB*, 270:471–480, 1997.

[59] L. Rychlewski, D. Fischer, and A. Elofsson. Livebench 6: Large-scale automated evaluation of protein structure prediction servers. *PROTEINS: Structure, Function, and Genetics*, 53:542–547, 2003.

[60] B. Schmidt, H. Schorder, and M. Schimmler. Massively parallel solutions for molecular sequence analysis. *First IEEE HiCOMB Workshop*, April 15 2002.

[61] H. P. Schwefel. Numerical optimization of computer models. *Wiley*, 1981.

[62] Bokhari S.H. and Sauer J.R. Sequence alignment on the cray mta-2. *Second IEEE HiCOMB workshop, Nice, France*, April 22 2003 2003.

[63] Y. Shan, G. Wang, and H. X. Zhou. Fold recognition and accurate query-template alignment by a combination of psi-blast and threading. *Proteins: Structure, Function, and Genetics*, 42(1):22–37, November 2001.

[64] K. Simons, R. Bonneau, I. Ruczinski, and D. Baker. Ab initio protein structure prediction of casp 3 targets using rosetta. *Proteins*, S3:171–176, 1999.

[65] M. J. Sippl. Recognition of errors in three-dimensional structures of proteins. *Proteins*, 17:355–362, 1993.

[66] J. Skolnick and D. Kihara. Defrosting the frozen approximation: Prospector - a new approach to threading. *Proteins: Structure, Function, and Genetics*, 42(3):319–331, February 2001.

[67] M. Snir, S. Otto, H.S. Lederman, D. Walker, and J. Dongarra. *MPI - The Complete Reference, Volume 1, The MPI Core, Second Edition.* The MIT Press, 1998.

[68] I. Sommer, A. Zien, N. V. Ohsen, R. Zimmer, and T. Lengauer. Confidence measures for fold recognition. *Bioinformatics*, 18(6):802–12, Jun 2002.

[69] Jones D. T., Taylor W. R., and J.M. Thornton. A new approach to protein fold recognition. *Nature*, 358:86–92, 1992.

[70] W. R. Taylor. Multiple sequence threading: an analysis of alignment quality and stability. *J Mol Biol.*, 269(5):902–43, June 1997.

[71] R. Thiele, R. Zimmer, and T. Lengauer. Protein threading by recursive dynamic programming. *J Mol Biol.*, 290(3):757–79, Jul 1999.

[72] S. Thomas and N. M. Amato. Parallel protein folding with stapl. *Third IEEE HiCOMB Workshop*, April 26 2004.

[73] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *Journal of Molecule Biology*, 231:75–81, 1993.

[74] Kumar V., Grama A., Gupta A., and Karypas G. Introduction to parallel computing, design and analysis of algorithm. *The Benjamin/ Cummings Publishing Company Inc.*, 1994.

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

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

[77] J. V. White and I. Muchnik. Modeling protein cores with markov random fields. *Math Bioscience*, 124(2):149–79, 1994.

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

[79] X. Xiao, E. Dow, R. Ebrhart, Z. B. Miled, and R. J. Oppelt. Gene clustering using self organizing maps and particle swam optimization. April 22 2003.

[80] D. Xu, O. H. Crawford, P. F. LoCascio, and Y. Xu. Application of prospect in casp4: Characterizing protein structures with new folds. *Proteins: Structure, Function, and Genetics*, 45(S5):140–148, 2001.

[81] J. Xu. *Protein structure prediction by Linear Programming.* PhD thesis, University of Waterloo, August 2003.

[82] J. Xu. Speedup lp approach to protein threading via graph reduction. In G. Benson and R. Page, editors, *Algorithms and Bioinformatics: 3rd International Workshop (WABI)*, volume 2812 of *Lecture Notes in Bioinformatics*, pages 374–388, Budapest, Hungary, September 2003. Springer.

[83] 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.

[84] J. Xu, M. Li, G. Lin, D. Kim, and Y. Xu. Protein threading by linear programming. *Pacific Symposium in Biocomputing (PSB) World Scientific*, pages 264–275, January 2003.

[85] Y. Xu and D. Xu. Protein threading using prospect: design and evaluation. *Proteins*, 40(3):343–54, Aug 2000.

[86] Y. Xu, D. Xu, O. H. Crawford, F. Larimer, E. Uberbacher, M. A. Unseren, and G. Zhang. Protein threading by prospect: a prediction experiment in casp3. *Protein Eng.*, 12(11):899–907, Nov 1999.

[87] Y. Xu, D. Xu, and E. C. Uberbacher. A new method for modeling and solving the protein fold recognition problem. In *Proceedings of the second annual international conference on Computational molecular biology*, pages 285–292, 1998.

[88] J. Yadgari, A. Amir, and R. Unger. Genetic threading. *Constraints*, 2-3(6):271–292, 2000.

[89] N. Yanev and R. Andonov. Solving the protein threading problem in parallel. *Parallel and Distributed Processing Symposium 2003 Proceedings International*, page 8, April 2003.

[90] N. Yanev and R. Andonov. Parallel divide and conquer approach for the protein threading problem. *Concurrency and Computation: Practice and Experience*, 16(9):961–974, 2004.

[91] N. Yanev and R. Andonovy. The protein threading problem is in p? *Rapport de recherche de l'INRIA-Rennes, Equipe : SYMBIOSE No. 4577*, 3:15, October 2002.

[92] T.K. Yap, P.J. Munson, O. Frieder, and R.L. Martino. Parallel multiple sequence alignment using speculative computation. *Proceedings of the 1995 International Conference on Parallel Processing*, August 1995.

[93] T.K. Yap, P.J. Munson, O. Frieder, and R.L. Martino. Parallel multiple sequence alignment using speculative computation. *Proceedings of the 1995 International Conference on Parallel Processing*, 9(3), March 1998.

[94] A. Zemla, C. Venclovas, J. Moult, and K. Fidelis. Processing and evaluation of predictions in casp4. *PROTEINS: Structure, Function, and Genetics*, 5:13–21, 2001.

[95] C. Zhang and S. Kim. Environment-dependent residue contact energies for proteins. *PNAS*, 97(6):2550–2555, 2000.

[96] X. Zhang, D. Waltz, and J. P. Mesirov. Protein structure prediction by a data-level parallel algorithm. In *Supercomputing '89: Proceedings of the 1989 ACM/IEEE conference on Supercomputing*, pages 215–223, New York, NY, USA, 1989. ACM Press.

Vita Auctoris

The author, Md. Rafiqul Islam was born in Sirajganj, Bangladesh. He did B. Sc. in Civil Engineering degree from Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh. He had been working in Bangladesh and in Zambia as a consultant before he immigrated in Canada in 2000.

He completed his B. Sc. in computer science (with distinction) from University of Windsor in 2002. He is currently a candidate for the M. Sc. in computer science, supervised by Dr. Alioune Ngom, at the University of Windsor, Ontario and hopes to graduate in Summer 2005. His research interest includes to apply machine learning and high performance computing in the field of bioinformatics.