12-2007

Protein Threading for Genome-Scale Structural Analysis

Kyle P. Ellrott
University of Tennessee - Knoxville

To the Graduate Council:

I am submitting herewith a dissertation written by Kyle P. Ellrott entitled "Protein Threading for Genome-Scale Structural Analysis." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Life Sciences.

Ying Xu, Major Professor

We have read this dissertation and recommend its acceptance:

Robert L. Hettich, Victor Olman, Hong Guo, Elizabeth Howell

Accepted for the Council:

Carolyn R. Hodges

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

To the Graduate Council:

I am submitting herewith a dissertation written by Kyle P. Ellrott entitled "Protein Threading for Genome-Scale Structural Analysis". I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Life Science.

Dr. Ying Xu, Major Professor

We have read this dissertation
and recommend its acceptance:

Dr. Robert L. Hettich

Dr. Victor Olman

Dr. Hong Guo

Dr. Elizabeth Howell

Accepted for the Council:

Carolyn R. Hodges, Vice Provost and
Dean of the GraduateSchool

Protein Threading for Genome-Scale Structural Analysis

A Dissertation

Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Kyle P. Ellrott

December 2007

Dedication

To my wife, who gave me her real phone number when I asked.

To Jared, because.

To Travis, just to annoy him.

To my grandfather, who lived just long enough to see me add a 'Dr.' to my name.

To my parents, who tell me that when I was born they looked at me and said, "Lets see how we can screw this one up".

To God, who doesn't get the credit he deserves.

Acknowledgments

I would like to thank Dr. Ying Xu for his supervision, Dr. Jun-Tao Guo for his guidance, Dr. Victor Olman for explanations, Joan Yantko for her organization, and Irina Ellrott for her grammar.

Also, I would like to thank the members of my dissertation committee; Dr. Ying Xu, Dr. Victor Olman, Dr. Hong Guo, Dr. Robert Hettich, and Dr. Elizabeth Howell.

In addition, I would like to thank various members of the Computational System Biology Lab at the University of Georiga, including Dr. Phuongan Dam, Dr. Hongwei Wu, Dr. Claire Gervais, Dr. Fenglou Mao, and Dr. Guojun Li.

The work is, in part, supported by

- The National Institutes of Health (R01 AG18927)
- The National Science Foundation (DBI-0354771/ITR-IIS-0407204/CCF-0621700)
- A Distinguished Cancer Scholar grant from the Georgia Cancer Coalition

- The Office of Biological and Environmental Research, US Department of Energy, under Contract DE/FG-2-04ER63714,

- The US Departments of Energy's Genomes to Life program (www.doegenomestolife.org), under the project 'Carbon Sequestration in Synecococccus sp.: from Molecular Machines to Hierarchical Modeling'

- The Office of Health and Environmental Research, US Department of Energy, under Contract No. DE-AC05-000R22725

- National Science Foundation Grants (DBI-0726924/DBI-0542119004)

I would also like to thank the San Diego Supercomputer Center and the National Science Foundation for providing access to their supercomputer resources in support of this work.

Abstract

Protein structure prediction is a necessary tool in the field of bioinformatic analysis. It is a non-trivial process that can add a great deal of information to a genome annotation. This dissertation deals with protein structure prediction through the technique of protein fold recognition and outlines several strategies for the improvement of protein threading techniques. In order to improve protein threading performance, this dissertation begins with an outline of sequence/structure alignment energy functions. A technique called Violated Inequality Minimization is used to quickly adapt to the changing energy landscape as new energy functions are added. To continue the improvement of alignment accuracy and fold recognition, new formulations of energy functions are used for the creation of the sequence/structure alignment. These energies include a formulation of a gap penalty which is dependent on sequence characteristics different from the traditional constant penalty. Another proposed energy is dependent on conserved structural patterns found during threading. These structural patterns have been employed to refine the sequence/structure

alignment in my research. The section on Linear Programming Algorithm for protein structure alignment deals with the optimization of an alignment using additional residue-pair energy functions. In the original version of the model, all cores had to be aligned to the target sequence. Our research outlines an expansion of the original threading model which allows for a more flexible alignment by allowing core deletions. Aside from improvements in fold recognition and alignment accuracy, there is also a need to ensure that these techniques can scale for the computational demands of genome level structure prediction. A heuristic decision making processes has been designed to automate the classification and preparation of proteins for prediction. A graph analysis has been applied to the integration of different tools involved in the pipeline. Analysis of the data dependency graph allows for automatic parallelization of genome structure prediction. These different contributions help to improve the overall performance of protein threading and help distribute computations across a large set of computers to help make genome scale protein structure prediction practically feasible.

Contents

1 Introduction **1**

1.1 History of Protein Threading 7

1.2 Overview of Protein Threading Process 10

1.2.1 Energy Functions 11

1.2.2 Algorithms 12

1.2.3 Fold Recognition 16

1.3 Applications of Protein Threading 17

1.4 Current Research 21

2 Threading Energy Functions **25**

2.1 Introduction 26

2.1.1 Energy Functions 26

2.1.2 Violated Inequality Minimization 28

2.2 Materials and Methods 31

2.2.1 Violations 35

2.2.2 Violation Minimization 37
2.3 Results 40
2.4 Discussion 42
2.5 Conclusions 42

3 Variable Deletion Energies **44**

3.1 Introduction 44
3.2 Methods 48
3.3 Results 60
3.4 Discussion 67
3.5 Conclusion 69

4 Conserved Substructure Analysis for Threading Refinement **70**

4.1 Introduction 70
4.2 Methods 73
4.2.1 Fragment Selection 75
4.2.2 Testing and Training 76
4.3 Results 77
4.4 Discussion 78
4.5 Conclusion 79

5 Integer Programming Based Threading **80**

5.1 Introduction 81

5.2 Methods 88
5.2.1 Integer Programming 88
5.2.2 Energy Functions 90
5.2.3 The Deletion State 91
5.2.4 Terminal Deletions 92
5.3 Results 94
5.4 Discussions 98
5.5 Conclusion 99

6 A Genome Scale Protein Structure Prediction Pipeline Using Automatic Parallelization 100

6.1 Introduction 101
6.2 Materials and Methods 105
6.2.1 Pipeline Architecture 105
6.2.2 Pipeline Description 109
6.2.3 Algorithms and Data Structures 111
6.2.4 Programming Environment 113
6.2.5 Parallel Aspects of Data Transfer 114
6.2.6 Scheduling Parallel Tasks 117
6.2.7 Example Usage of Pipeline Programming 119
6.3 Results 122
6.3.1 Genome Applications 122

6.4 Discussion 123

6.5 Conclusion 125

7 Conclusions and Perspectives **127**

Bibliography **133**

Appendix **146**

Vita **149**

List of Tables

2.1 Sample energy patterns that can be used for Violated Inequality Minimization 37

2.2 Row Differences for energy vectors in 2.1 37

2.3 Top 1,5 scores over the course of weight training 40

3.1 A comparison using different gap function. 61

3.2 Side by side comparison of old method and IFA 63

3.3 Zscore based fold recognition results. 64

3.4 Gradient Boosting based fold recognition results. 64

3.5 The Correlation coefficients of the gap energies, as applied to the two threading method results. 68

4.1 A comparison using different gap function. 78

4.2 Side by side comparison of the Original Alignments and the Structure Refined Alignment 79

6.1 AA_{pred} tested Performance 123

6.2 Protein Structure Prediction Pipeline Results 124

List of Figures

1.1 Historical Timeline of Protein Threading Algorithms 12

2.1 Violation count over time using Static VIM training 41

3.1 IFA information associated with the SCOP identifier 'd1qhoa2'. . . . 51

3.2 A sample multiple sequence alignment used to calculate B_{del}. 53

3.3 The conversion from the sequence based model used to represent alignments, typically as outputted by Blast. 54

3.4 The running averages of the two methods. 61

3.5 An example alignment between SCOP domains 'd1f1za1' and 'd1ucra_'. 62

3.6 Sensitivity and Specificity for Fold Level pairs 65

3.7 Sensitivity and Specificity for Super Family Level 66

3.8 Sensitivity and Specificity for Family Level 67

4.1 An example of a section of a target protein sequence that has been aligned to two similar protein sub structures 72

4.2 An example re-threading of the target protein d1hg3a_ to template d1vc4a_ 77

5.1 An example of template cores being aligned into a target sequence . 84

5.2 An example where cores 1 and 4 form a conserved substructure. . . . 85

5.3 The connection graph shows which cores have active pairwise energies. 86

5.4 The growth of the number of interactions as the number of cores increases. 89

5.5 Core B, aligned to the same position: in the regular state and in the deleted state. 91

5.6 Core deletion, as seen in an alignment matrix 93

5.7 An example protein structure alignment of d1cwva4 and d1cdy_1. . . 95

5.8 An example of an alignment created by the old method, and one created by the core deletion model. 95

5.9 The alignment between TM1457 and PMS2 [Fischer et al., 2003]. . . 97

6.1 An Example of Operational Parallelism 112

6.2 The expansion and contraction of parallel tasks 118

6.3 The expansion and contraction of parallel tasks spread across time . 119

6.4 Computational Performance of Mandelbrot calculation vs. Cluster Size 120

6.5 The data parallel nature of the protein structure prediction problem 121

6.6 The data parallel portions of Protein Structure prediction, as viewed on the pipeline workflow graph 122

Chapter 1

Introduction

A protein is a chain of residues, each of which is from one of the twenty amino acids that occur in living organisms. This chain folds into a packed three-dimensional structure that is responsible for the function of that protein. Proteins are one of the major players in the biochemical reactions in living organisms. An understanding of their structures provides insight into the basic biochemical interplay and the complex chemical networks that are necessary to maintain life. The amino acid sequence of a protein can be determined from the DNA sequence that encodes it. Modern sequencing techniques, along with bioinformatic techniques, can efficiently determine the amino acid sequences of the number of genes encoded in a genome in a short amount of time. The protein structure, on the other hand, is significantly more challenging to determine. Protein structures can be determined experimentally using X-ray crystallography or nuclear magnetic resonance (NMR) spectroscopy methods. These efforts may require several months for protein expression and purification and

do not always necessarily yield fruitful results. It is this situation, a wealth of amino acid sequences and a lack of efficient and effective techniques for solving their protein structures, that has pushed forward the field of protein structure prediction.

Elements of protein structure can be divided into four groups; primary structure, secondary structure, tertiary structure, and quaternary structure. Primary structure is the sequence of amino acids as they have been linked together by the poly-peptide chain. Secondary structure deals with the hydrogen bonds that form between non-neighboring residues. These secondary structure relationships can be roughly classified into one of three groups: loops/coils, α-helix, and β-strands. When classified using this system, secondary structure features can be predicted with 85% accuracy by a trained Neural Network [Jones, 1999]. Tertiary structure is the global structure of a protein that is formed by the secondary structure elements interacting in a three dimensional space. Quaternary structure reflects the physical interactions of several individual protein chains packed into a single unit.

One of the key observations in the proteomics field is that an amino acid sequence uniquely determines its protein structure, which in turn determines the function of the protein. From a biological point of view, it is not the amino acid sequence of a protein that is important. Instead, the function of the protein is what determines how it is associated with the biological pathways in an organism. There are many cases of proteins that have dissimilar sequences with similar structures. One example is TIM (triose-phosphate isomerase) barrels. If a structural similarity is able to be determined, even without sequence similarity, the function of an unknown gene

could be inferred. So beyond the preliminary goal of simply trying to determine the structural characteristics of a protein, there is the possibility of functional analysis, and thus it provides a new source of information for biological investigation. With the growing number of genomes and genes, this give an extra source of data for genome analysis and annotation.

Protein structure prediction has been evolving since the 1970s when Anfinsen [Anfinsen, 1973] demonstrated that nearly all the information a protein needs in order to fold is stored in its amino acid sequence. This means that given the string of amino acids, one could theoretically reconstruct the structure of the protein. Although the technique is theoretically possible, it is not easy. Folding, the process by which an amino acid chain folds into a compressed and energetically favorable structural conformation, is the result of atomic interactions and the force to reduce surface area of hydrophobic amino acids that are in contact with the water molecules that surround the protein. Computationally this interaction is incredibly challenging to simulate.

There are three classes of methods currently used for protein structure prediction: i) Homology modeling ii) Protein Threading iii) *Ab-Initio* techniques. The first two methods are categorized as comparative modeling techniques. Comparative modeling relies on using previously determined protein structures to bootstrap protein structure predictions of new amino acid sequences. Homology modeling is the method aligning a new protein sequence to an existing protein structure through

sequence comparison. Homology modeling is most effective when there is a high degree of sequence homology. Protein threading is an expansion of this idea. Rather than just relying on sequence comparison to find structural similarity, the implications of amino acid placement with respect to the template protein structure are considered. In cases where there is little sequence identity, protein threading is able to perform better by matching folds that homology modeling is unable to find.

In comparative modeling, the energy is a preference measure of placing a residue at a particular position on the template sequence. Beyond the comparative modeling there are atom level prediction approaches. These techniques include quantum calculations, molecular dynamics simulations, and Monte Carlo sampling. These techniques treat the protein as a large molecule and seek placing the atomic coordinates of the protein's amino acids by minimizing the physical interaction energies between them as well as between them and the solvent. This type of modeling is called *ab-initio* folding [Hardin et al., 2002, Ishida et al., 2003], which means "from the first principles of physics", because it relies on the basic laws of physics without additional assumptions or special models. These methods are built from the ground up without information derived from solved protein structures. For these techniques, the energy functions being minimized are representative of the actual atomic energies, and not statistically derived as the ones used in comparative modeling. It should be noted that these modeling methods require a significant amount of computational power, much beyond what is available by existing computing power, and their application has been limited to smaller proteins. While they do not take

advantage known protein structures, they are able to work, theoretically speaking, even when there is no known homologous structure in the protein structure database.

In recent years there has been a certain amount of blending of these types of the protein threading techniques and *ab-initio* methods. New ideas such as pseudo-*ab-initio* modeling, or mini-threading, attempt to rebuild a protein by matching short spans of an amino acid chain with commonly found fragments [Rohl et al., 2004, Jones, 1997]. This has the advantage of utilizing known structural motifs to constrain the search space and substantially reduce the required computing time by *ab-initio* techniques. These structure fragments will then be assembled, typically using Monte-Carlo sampling. As with *ab-initio* techniques, pseudo-*ab-initio* structure prediction is most effective when there are no matching folds in the structure database. If there are matching folds in the database, then comparative modeling is a more effective method.

Comparative modeling techniques can only work when the protein structure database is diverse enough to contain a large variety of protein structures. A survey of all proteins structures that have been solved reveals that there are a limited number of stable structural fold families. The majority of random amino acid sequences do not produce stable soluble proteins. Studies have shown that expression and proper solubility of a random amino acid sequence are unlikely. In one such study, only 20% of the library of randomly generated amino acid sequences were expressed in detectable quantities and only 20% of those expressed proteins were soluble in cell lysate [Prijambada et al., 1996]. When looking at the statistics of new protein

structures that have been solved and stored in the PDB database [Berman et al., 2000], the central repository site of solved protein structures, there has also been a noted decline in the percentage of new folds vs the total number of folds being submitted. These signs indicate that there is a limited number of protein folds that occur in nature.

No single technique can handle all protein structure predictions. Instead there are a variety of methods that allow us to approach different portions of the spectrum of possible sequences. Homology modeling is best when a match to a known structure can be found with simple sequence alignment. Moving further away, protein threading can find structural matches in cases where there is little sequence homology. For cases where there are no structural matches in the structure database, psuedo-*ab-initio* techniques can be applied. And finally, for small proteins, one can apply *ab-initio* techniques such as molecular dynamics for their structure prediction.

The research in this dissertation is primarily concerned with the advancement of the techniques related to protein threading. This dissertation seeks to improve the energy functions and algorithms related to protein threading, as well as fitting protein threading in the larger framework of a collection of tools that help predict structural features of proteins.

1.1 History of Protein Threading

There are three main ideas that guide the concepts of protein threading. First, proteins with similar sequences will adopt similar structures. Secondly, there are many examples of unrelated sequences adopting similar folds. And finally, there is only a relatively small number of possible fold structures that a protein sequence can adopt.

The first papers to propose the idea of protein threading was the work by Bowie at al. [Bowie et al., 1991] in 1991. This technique was first referred to as 'threading' in a the paper by Jones at al [Jones et al., 1992]. The concept of protein threading differentiates itself from simple homology modeling in that it attempts to take into account structural features of template structures.

The method of protein threading is comprised of several underlying techniques and processes. In order to develop a protein threading method, one needs to concentrate on the following areas: the energy functions, structural template library selection, the threading algorithm, and the fold recognition techniques.

One of the key features of protein threading is the analysis of the spacial relationships between residues as they are placed in the template protein structures. Although two residues may not be next to each other in the linear sequence of the protein chain, due to the folding of the protein they may end up spatially close to each other. The preferred interactions between amino acid types can be mapped in

a distance dependent multi body interaction form. Typically multi body interactions are represented as pairwise terms. This energy can be described with the two amino acid types, and the distance between them, typically measured from the C_β atom to the C_β atom of the two residues involved. If two residues tend to repel each other or are too large to be packed within a small radius, a close proximity will be unfavorable. This type of information will provide useful information that can guide to proper alignment between the sequence of a target protein and the structure of a template to find the correct folds among a database of protein structures.

However, it is worth noting that the problem of optimizing an alignment, allowing gaps and pairwise interaction energy is shown to be NP hard [Lathrop, 1994]. The creation of the optimization algorithm is what allows a proper alignment to be calculated that minimizes the energy function. The simplest example of an alignment algorithm is the Smith-Waterman alignment algorithm. The type of optimization algorithm that is used affects the type of energy functions that can be employed during the sequence optimization. Smith-Waterman algorithm is not capable of optimizing a threading alignment when non-local residue interactions are considered in the total energy function.

Beyond the alignment of a protein sequence to a structure, there is the need for correct fold recognition. A target sequence is threaded against the templates in a representative set of proteins. Once this has been done, the characteristics of the alignment of the target to each of the templates are assessed in order to determine which alignment is the optimal. This is typically accomplished by training

machine learning techniques to recognize the correct fold. Previous research has used techniques such as neural networks [Xu et al., 2002], SVMs [Xu, 2005], and gradient boosting [Jiao et al., 2006] based functions for fold recognition.

Because of the importance of protein structure prediction, the Critical Assessment of techniques for protein Structure Prediction (CASP) competition was formed to allow different techniques to perform in a blind scenario. First started by John Moult and others in 1994 and held every two years, the contest begins with a set of proteins that have had their structures determined by experimentalists. These structures, determined by X-Ray crystallography and NMR, are withheld from public release. Teams of structure predictors are given the amino acid sequences of these unreleased structures so that they may attempt predictions. These predictions are judged and the results are tallied so that the best methods can be determined. CASP allows for a blind test of the predictive abilities of the variety of protein structure techniques that are being developed. It is a way to objectively measure the progress of the field of protein structure prediction.

In addition to CASP, there is CAFASP (Critical Assessment of Fully Automated Structure Prediction), which was initiated in 1998 after CASP3. This contest seeks to test the abilities of fully automated systems to predict protein structures without the intervention of humans. In this test, the sequences are submitted to a contestant's website for automated prediction. The results are due within a shorter period of time than what is allowed in the manual CASP prediction contest, typically within 24 hours.

CASP has grown in popularity and importance in the protein structure prediction community. During CASP1 there were only 35 groups that where part of the experiment. By 2004, over 200 prediction teams from 24 countries participated in CASP6. In the fold recognition category, only 9 groups participated in CASP1, but by CASP6 that number had increased to 165.

1.2 Overview of Protein Threading Process

The process of protein threading can be broken into several key stages. Each stage is subject to independent research and refinement.

- Identification of Structural Templates. This is the process of building a representative list of protein structures. Each template in the list will be used to thread against query or target sequences. The representative list needs to cover all possible known fold types, but at the same time not over represent any one family. One example of a representative list creation tool would be PISCES [Wang and Dunbrack, 2003]

- Alignment of the query sequences with template structures. This can be accomplished with a threading technique such as PROSPECT [Xu et al., 1998]. This process will be more fully explored in the following chapters of this dissertation.

- Build a model for the query sequence. This process includes core modeling, side chain modeling, and loop modeling based on the structural based on

the structural restraints discovered by threading. Tools that do these sort of calculations include Modeller [Sali and Blundell, 1993] and Jackal [Xiang and Honig, 2001].

- Model Evaluation. At this point, the physical properties of the generate model are evaluated to make sure they obey known physical characteristics. Tools that can accomplish this task include PROCHECK [Laskowski et al., 1993] and 'What If' [Vriend, 1990].

1.2.1 Energy Functions

In protein threading the alignment between a target sequence and a template structure is optimized over a set of statically derived energy functions. These energy functions need to be able to distinguish correct alignments from the incorrect ones. Energy functions map simple amino acid homology as well as interactions with more complex structural features. These energies will be more thoroughly examined in Chapter 2. The main idea to understand is how the complexity of an energy function effects the type of optimization algorithm that can be used to solve the sequence-structure alignment problem. We are primarily concerned with two types of energy functions as classified by the number of residues that they are dependent upon. Single residue energies are only dependent on the alignment of one amino acid in the target sequence to one residue in the protein template structure. Residue pair energies are dependent on the simultaneous alignment of two structural positions to two of the target sequence amino acids. The residue pair interaction energies

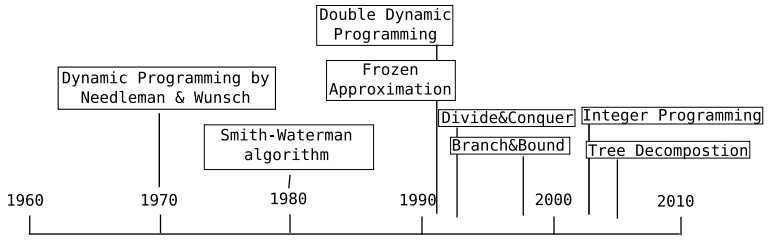

Figure 1.1: Historical Timeline of Protein Threading Algorithms

attempt to capture the interaction of two residues situated in a three dimensional structure as they are replaced with two new amino acids. Residue pair energies will often be referred to as pairwise energies.

1.2.2 Algorithms

The main idea behind protein threading is the incorporation of non-local energies. Residue pair energies, for example, are energies that are calculated from the placement of two non-adjacent amino acids to two spatially close structural positions. It has been shown that the problem of optimizing of an alignment, allowing gaps and pairwise interactions is considered to be NP hard [Lathrop, 1994]. This means that for large proteins, the computation resources needed in order to calculate optimal alignments are simply intractable. Because of its similarity to homology modeling, the core algorithmic ideas behind protein threading are very similar. However, there have been more complex methods suggested in order to accommodate more complex definitions of amino acid energies. The major algorithmic ideas have been listed, and outlined in Figure 1.1.

- Dynamic Programming by Needleman & Wunsch [Needleman and Wunsch, 1970] and Dynamic Programming by Smith & Waterman [Smith and Waterman, 1981]

  The first two algorithms, Dynamic Programming by Needleman & Wunsch and Smith & Waterman were originally designed for sequence level alignment, i.e. homology modeling. By the nature of dynamic programming, these algorithms cannot handle non-local interactions. The optimal alignment returned by these algorithms, are not guaranteed to be optimal in regards to objective functions that include two-body interaction energies.

- Double Dynamic Programming [Jones et al., 1992]

  This is one of the first heuristic methods to create an alignment with respect to the residue pair interaction in the template protein. This method employs a two level dynamic programming matrix. For each cell in the high level matrix the likelihood is calculated by another low level matrix that is derived with the constraint of the residues in the parent cell being aligned.

- Frozen Approximation

  Frozen approximations attempt to deal with residue pair energies by doing local calculations with residue pairs calculated from template structure, not using the residues from the target sequence in the pair position. Algorithmically, this is the same as the dynamic programming methods, but with a new energy model.

- Branch and Bound [Lathrop and Smith, 1994]

  Branch and Bound in an optimization technique that breaks the problem into a tree. Branches of the problem are explored while the best scores of the previously explored regions are stored. If a branch is viewed to be so bad that none of its children could redeem the score to obtain a minimum, a boundary is reached. Further exploration of the branch is not needed. Because the bounding of the tree is dependent on the structure of the tree and the energy functions used, there is no guaranteed boundary to the time needed to solve the problem.

- Divide and Conquer [Xu et al., 1998]

  Divide and Conquer is an algorithmic technique that introduces the concept of a core. A core is a secondary structure element that interacts with at least one other secondary structure element. It is only in between cores that residue pair energies interactions are calculated. Deletions and insertions are not allowed inside of cores, and for the rest of the protein, the loop regions, the residue pair energy interactions are not considered. However, during the divide portion of the algorithm, neighboring cores were represented by possible interaction tables. These tables represent the possible amino acids a neighboring residue could be, prior to that decision being made. This meant that the algorithm would be bound by the number of configurations that table was able to represent. There was one state for every amino acid, which meant that

the pair energy term could only describe specific amino acid relationships. More complex relationships, such as mutation profile or distance dependent relationship, could not be efficiently mapped to the neighbor table. In this way, the algorithm limited the types of energy functions that could be used.

- Integer Programming [Xu et al., 2003]

  Integer programming reapplies the concept of the core, however this time a combinatorial technique from operations research is applied to optimize the alignment. With this technique, each and every neighbor interaction is mapped explicitly and considered during optimization. This means that integer programming can handle more complex energy interactions, such as those based distance dependence or mutation profile.

- Tree Decomposition [Xu et al., 2005]

  Tree decomposition utilizes methods from graph theory that allow for deconstruction of the interaction graph to determine an order of dynamic programming that can still allow for the inclusion of residue pair energies. It works efficiently, however the method is memory bound by the number of allowed core positions. Thus it is not able to consider all possible core placements in an alignment. If a core's position in the target sequence is not in the list of possible candidate sites, then the algorithm will fail to find the optimal alignment.

1.2.3 Fold Recognition

Once a target sequence has been threaded against a protein structure library, one needs to select the correct template. Every one of the alignments between the target and a particular template is optimal given the energy functions used, however, that does not mean that the alignment is the correct one. Most alignments are meaningless, and the template structure has very little in common with the native protein structure of the target sequence. However, it is necessary to be able to discern when this is the case. This is the problem of protein fold recognition.

One common technique used is to judge a fold to be correct or not is by its Zscore. A Zscore is a measurement of statistical significance which is used to compare the energy of the optimal alignment, E_{min} of the target sequence to other proteins of similar amino acid composition against the same protein template. This can be measured by shuffling the protein sequence randomly and re-threading multiple times. The energy distribution is sampled to find the mean, μ and standard deviation σ. The Zscore is defined to be the $\frac{E_{min}-\mu}{\sigma}$.

Beyond statistical analysis, there are heuristic techniques used to recognize energy and feature patterns common to the native structural folds. In this method fold recognition is achieved by using a machine learning technique and training it against a set of representative protein pairs, a set of sequence along with their native structural folds. To train these systems, a set of characteristics derived from the sequence-structure alignment is used as the input features. These reflect the

summed values of various energy components, the number of aligned residues, the comparative lengths of the sequences being aligned, and the Zscore. The features that are trained against typically include: Target Size, Template Size, Number of Aligned Residues, Alignment Sequence Identity, Total alignment score, The individual scores, Zscore. These different features form an input vector to a machine learning system. The response variable is typically the number of correctly aligned residues as compared to a protein structure/structure alignment done by Sarf [Alexandrov, 1996], FAST [Zhu and Weng, 2005], Lock2 [Shapiro and Brutlag, 2004].

The machine learning techniques that have been employed include Support Vector Machines (SVMs), Neural Networks, and Gradient Boosting techniques. Neural networks are modeled after neural synapse activities. A set of cascading sinusoidal functions are trained to approximate the target function. SVMs attempt to use a set of sample points to describe the boundary between classified regions. Gradient boosting uses a greedy algorithm to complex function regression.

1.3 Applications of Protein Threading

While protein threading is a fascinating computational problem, it is its application in the real world situations that will decide whether or not the technique has value. The best evidence for success of protein threading is demonstrated by studies where the structure has been threaded without any confirmed structural information about

the original sequence and that prediction has been subsequently verified by an experimental structural determination. This of course is an ideal situation that renders the original protein threading experiment obsolete. Situations where experimental structural verification is inconvenient, or even impossible, are ideal scenarios for the application of protein threading. To truly prove it's worth, protein threading should be used as a method to formulate a hypothesis or too narrow down the search space of possible candidates in an experiment. In these experiments, protein threading provides a guide for experimentation, even though its results are never verified with a protein structure determined by X-ray crystallography.

Threading needs to be verified using multiple techniques. These experiments can be either direct or indirect in their validation of the protein threading results.

Testing methods that provide direct validations are best defined by the CASP experiment. It is a perfect example of method for proving the ability for protein threading techniques to identify common folds. Although the structures used in CASP have been experimentally determined, from the point of view of the experimenter doing the protein threading, they are unknown. But this can be seen as "low risk" in that the structure has already been determined, and no critical biological data will be derived from the computational experiment.

Experiments that show protein threading can be used for scientific discovery include work done by Edwards et al [Edwards and Perkins, 1996]. They predicted the fold type of the Willebrand Factor type A domain without sequence or functional similarity to the template protein. It was only later that the structure was

determined and the fold recognition shown to be accurate. In a similar experiment, Dong Xu et al [Xu et al., 2001a], utilized protein threading techniques to predict the structure 3 domains of vitronectin. This experiment provided indirect verification of protein threading results because the predicted structures matched information derived to protease-sensitivity studies, and information that was available about the different di-sulfide bonds.

Demonstrating that these techniques are widely adaptable, namely in the inverse protein threading problem of de-novo design, is the work of Kuhlman et al [Kuhlman et al., 2003] who where able to design a target protein using the Rossetta library. The resulting protein sequence was grown and crystallized. The structure solution of the actual protein showed that it was within 1.2 angstroms of the original designed structure.

Beyond the simple set of prediction and verification experiments, there are other biological experiments that demonstrate protein threading's ability to help provide important information. In these experiments protein threading revealed new information, even though the predicted structure was never experimentally determined. Instead, protein threading helps to formulate a hypotheses about structural elements, such as binding site conformations or residue interaction properties.

Some researchers utilize protein threading as a method to identify conserved residues in families of proteins. Wong et al. [Wong et al., 2001] used a structural homologue of the *Pseudomonas aeruginosa* protein OprM to create a model. This model was used as a template to analyze insertion and deletion mutants. This

information allowed them to predict areas of the protein structure where sequence changes would have less effect on substrate specificity. Work by Saleem et al [Saleem et al., 2004] utilized protein threading to identify potential sites for point mutation analysis in the forkhead domain of the FOXC1 transcription factor. Using protein threading information, and other sources of data, this group was narrowed down the search space to a set of 6 amino acids, three of which where shown to have a large effect on the functionality of FOXC1. Mueller et al [Mueller et al., 2004] utilized protein threading to model 28 structures of predicted seminal fluid proteins. These structures where analyzed to find residues associated with important structural and functional properties. Given this information, they where able to show the conserved structural features in this set of proteins, despite their rapid sequence mutation rate.

The protein threading energy functions have also been used to study binding specificity. Nese Kurt et al [Kurt et al., 2003], utilized protein threading methods to differentiate binding and non-binding sequences involved with HIV-1 protease. The HIV-1 protease, essential in the replication of the virus, is considered to be a major drug target, but unfortunately it has a large range of peptides that it can bind to. Contact and distance dependent statistical energies, similar to the two body energies in protein threading, were used by a algorithm similar to protein threading to determine possible peptide sequences that bound to the protease.

1.4 Current Research

There has already been several decades of research in the field of protein structure prediction. Protein threading is a deep subject with a large amount of previous research and ideas. It has been refined to the point that dramatic increases in performance are now very rare [Moult et al., 2007]. But just because a technique is well researched, it does not mean there is no more to be done with it. Rather than incremental refinements to the technique, we should begin to think of protein threading in a larger scale.

There are two different ways that we can begin to think 'big' when it comes to protein threading. First, we can start by thinking of how threading relates to the larger ecology of protein sequence/structure analysis tools. There are many different tools that provide lots of different information. This information can be used to improve protein threading results. In addition, the collective results of a protein threading experiment could be a valuable source of information. If analyzed these results could be used to refine the use of the protein threading technique.

The second way to think 'big' about protein threading is to study all possible protein structures encoded by a genome rather than looking at individual proteins. This effort could help connect protein structure prediction to the growing field of systems biology, where the goal is to understand the complex interconnected systems that help an organism to survive, rather than just looking at the individual

components. To do this, protein threading has to be made available at a genome level.

This dissertation concentrates on protein structure prediction through the technique of threading. The newest *ab-initio* based techniques are able to determine protein structures of small proteins relatively successfully [Bradley et al., 2003]. However, these techniques are computationally intensive and work better then other techniques when there is no related structures already in the structure database. Because of these reasons, these techniques are less applicable to genome scale structure analysis. First, at the genome level, with *ab-initio* techniques the amount of computer power needed to generate all the protein structures would be enormous, although this is not intractable. But because these structures would be generated from fragments, there would be little to no existing fold annotation. If a protein is matched to an existing fold, it is very likely that its function can be inferred by other members of the same structural fold. Thus at the genome level, *ab-initio* based techniques would provide more structural information, but not necessarily more functional information.

This dissertation outlines my research in the area of protein structure prediction by protein threading, including improvements in heuristic energies and applications of new alignment algorithms. The initial chapters tackle the first proposed problem, of connecting protein threading to more sources of information. This begins with Chapter 3, where my research leads to an energy function that takes advantage of information gained by studying the family of a particular protein. This energy

function provides a new way of modeling the penalty involved in the insertion and deletion of residues during alignment. To further refine the use of protein threading, in Chapter 4, I have suggested a method by which protein threading results are analyzed in an attempt to find conserved substructures which can then be used to refine protein threading results. In this way, protein threading becomes a tool to inform the usage of protein threading.

In Chapter 5, I propose an algorithmic expansion to the Integer Programming protein threading method. The Integer Programming protein threading model is an established model with certain known limitations. By adjusting the model, I have demonstrated a way to remove some of those limitations.

Finally, in Chapter 6, I show how the technique of protein threading fits into the larger framework of techniques that can be applied to protein sequence/structure analysis. I outline a pipeline approach to structure prediction, where protein threading is a single tool in a large chain of tools. It is this large chain of tools that we call a workflow. The idea of a workflow is common to other sciences. I will demonstrate new ways to document and describe a workflow. We will show that with the application of graph theory to analyze the data dependencies, operations can be parallelized automatically.

The proposed workflow ideas also help to bring about the second proposed 'big' idea, moving to the genome scale. Because the techniques for automatic parallelization allow for protein structure prediction pipeline to run efficiently on large

computer cluster, the idea of scaling protein threading to genome levels becomes entirely feasible.

Chapter 2

Threading Energy Functions

Some of the text below has previously been submitted as:

Ellrott, K. Guo, J-T. Olman, V. and Xu, Y. "A Generalized Threading Model using Integer Programming with Secondary Structure Element Deletion", Genome Informatics, 17(2):248-258, 2006

Ellrott, K. Guo, J-T. Olman, V. and Xu, Y. "Improving the performance of protein threading using insertion/deletion frequency arrays" (Accepted Journal of Biochemistry and Computational Biology 2007)

Ellrott, K. Guo, J-T. Olman, V. and Xu, Y. "Improvement in Protein Sequence-Structure Alignment Using Insertion/Deletion Frequency Arrays". The proceedings of Computational Systems Biology 2007 (CSB2007), 335-342, 2007

2.1 Introduction

The first step in designing a protein threading method involves the judicious application of energy functions that will maximize the accuracy of residue alignments. This chapter focuses on available energy functions and how they operate. We will demonstrate how to integrate these separate energy functions into a cohesive protein threading objective function for alignment optimization.

2.1.1 Energy Functions

The core goal of a protein sequence structure alignment energy is to leverage the available information to create a correct alignment. In most cases, alignments are described as a mapping between two protein sequences or between two protein structures. In contrast, protein threading alignments are different because they are in an asymmetric situation. There is more information available about the template, whose structure has been elucidated, than there is about the target protein, where only the amino acid sequence is available. This asymmetry is handled with a new set of energy functions that are designed to utilize the available information.

There are two essential types of energy functions; point energy functions and pair energy functions. The point energy function evaluates the mapping of one residue of the target protein onto one of the residue positions in the template. Pair energy function evaluates the simultaneous placement of two target residues at two residue positions in the template. It is also possible to derive statistical energy functions

for three or four body interactions, but currently efficient global optimal algorithms for optimizing three or four body interactions do not exist. The energy functions we use can be classified into the following groups:

- Mutation Energy

  Mutation Energy is the statistical evaluation of the likelihood of one amino acid being replaced by another during the normal course of evolutionary mutation. Previously these statistics could be determined by a substitution matrix such as PAM [Dayhoff et al., 1978] or BLOSUM [Henikoff and Henikoff, 1992]. More recently it has been shown that substitution matrices derived from multiple sequence alignment form a more accurate quantitative description of mutation for a specific protein than matrices derived from global statistics.

- Singleton Energy

  Singleton Energy [Xu and Xu, 2000] describes a residue's propensity for existing in a certain type of environment. It primarily describes amino acids preferences for being exposed on surface areas of the protein versus being buried. Hydrophobic residues prefer to be buried while hydrophilic residues can exist on the surface of the protein with less energetic cost.

- Secondary Structure Energy

  Secondary Structure Energy deals with the alignment of the template's secondary structure and the predicted secondary structure of the target protein. Secondary structure prediction methods, such as PsiPred [Jones, 1999], have

already achieved almost 80% accuracy when predicting a three state possibility of a residue's assignment to a secondary structure.

- Multi-body interaction Energy

  Multi-body interaction energy describes the relationship of multiple residues as they behave in close proximity [Zhang and Kim, 2000, Zhang et al., 2004b]. While three and four body interactions exist, protein threading almost exclusively considers two-body interactions. These interactions are between the residues as a whole and not the specific atoms that comprise them. Typically calculations are based on the distance between the $C\alpha$ or $C\beta$ atoms of the two residues.

- Gap Energy

  Gap energies are the penalties for the insertions and deletions that are needed in order to generate an alignment between a target protein sequence and a template structure. Historically this has been viewed as a constant penalty for the removal or insertion of a single residue. However, our research has demonstrated that it pays off to form a more sophisticated model [Ellrott et al., 2007].

2.1.2 Violated Inequality Minimization

The energy functions for sequence-structure alignment are based an a variety of statistically derived functions. As a result these energy functions exist without units

to describe how to scale between each other. In order to place these function in the same equations, there must be scaling factors attached to them. In the formulation, there is a 'weight' assigned to each energy function.

Obtaining the set of weights that balance the different energies in such a way that when used for alignment they produce a correct results is a non-trivial problem. If the set of weights is not properly balanced, non-realistic results can occur. Thus the tuning of this array is critical for good performance. For N energy functions, one of the weights can be fixed to 1, creating an optimization problem for a $N-1$ dimensional optimization problem. Because these energy functions are derived from statistical analysis, there is no real way to integrate them to determine a slope for a search technique such as maximal gradient descent. The traditional methods for solving this problem are to use a grid search or a Monte Carlo search. In the grid search method, the $N-1$ dimension search space is searched in the regular fashion. The Monte Carlo search method attempts to optimize the W set by making random changes and measuring their effects.

Both grid search and Monte Carlo search have certain pros and cons when applied to the optimization of the W weight set. A grid search will test a large number of points that are nowhere near the optimal point, while a Monte Carlo search could become stuck in a local optimal if care is not taken. It is important to remember that simply testing a single set of parameters may take a large amount of time. To verify the threading performance accuracy determined by a set of weight parameters, every structure sequence pair in the training set must be threaded. While dynamic

programming based alignment is near instantaneous, the large number of possible pair interactions takes a while to calculate.

Because of the amount of time needed to sample a W set's performance, it is very important to minimize the number of sample points used to optimize the parameters. Because these parameters are dependent on the energy functions, it is necessary to retrain them every time the energy functions are changed. Minimizing the amount of time needed to optimize the W set's performance is critical if we are going to suggest new energy functions and test their effects on the prediction model.

The previously described methods of optimization are all designed for systems where the validity of each of the sample points can not be critically analyzed on the fly. In this particular training scenario, for every given sequence structure pair we know the true alignment. Thus, given a population of samples, we can describe which of the samples are accurate and which ones are not.

We have utilized a method called "Violated Inequality Minimization" (VIM) to adjust the weights. With this technique, energy profile differences between the predicted alignment and the known alignment are minimized by adjusting the weights. This method quickly adjusts the weights of the different parameters to account not only for their relative orders of magnitude but also their importance in generating a good alignment.

2.2 Materials and Methods

The specific formulation of the energies is derived from a variety of data sources and statistical samplings. We have formulated each of the energies as a function that relates to the alignment of a specific amino acids in the target sequence and a specific residue in the template protein.

Mutation Energy

$$E_m(i,j) = \sum_{a \in A} (aaprob(i,a)pssm(j,a)) \tag{2.1}$$

i refers to the ith amino acid position in the target sequence, while j is the jth residue position in the template structure. A is the set of the 20 amino acid types. $pssm(j,a)$ refers to the log score of amino acid type a at the jth position in the template, as returned by the MakeMat program in the NCBI toolkit. $aaprob(i,a)$ refers to the probability of an amino acid type a at position i in the target sequence, as retrieved from the checkpoint file generated by Psi-Blast [Altschul et al., 1997]. Both numbers are the result of Psi-Blast searches, which produces 20 element vectors for every amino acid in a sequence, where each element in the vector represents a score for each of the 20 amino acids. The difference is that pssm is in log space, while prob is a probability.

Singleton Energy

$$E_s(i,j) = \sum_a aaprob(i,a)F(j,a) \tag{2.2}$$

$$F(j,a) = -k_BTlog(\frac{N(aatype(j), ss_j, sol_j)}{N_E(j, ss_j, sol_j)}) \tag{2.3}$$

$$N_E(a, ss_j, sol) = \frac{N(a)N(ss)N(sol)}{N^2} \tag{2.4}$$

where j is the jth residue position in the template, and ss_j is the secondary structure of the jth residue, and sol_j is the solvent accessibility type of the jth residue. Using representative samples from the database of known proteins; $N(a)$, $N(ss)$, $N(sol)$ are the different amino acid types, the different secondary structure types, and the different solvent accessibility types, respectively. N is the number of sampled residues. $N(a, ss, sol)$ is the number of times when amino acid a was observed in secondary structure ss and solvent accessibility sol in the structural database that we used to derive our energy function, while N_E is the number of occurrences expected given the independent probabilities of the different factors.

Secondary Structure Match Energy

$$E_{ss}(i,j) = 3.0 - 10(ssprob(i, sstype(j))) \tag{2.5}$$

Where $ssprob(i,c)$ is the probability that the ith amino acid of the target belongs to secondary structure type c, as determined by a secondary structure prediction

program like PsiPred [Jones, 1999]. $sstype(j)$ is the secondary structure type of the template as determined by DSSP [Kabsch and Sander, 1983].

Residue Pair Energies

Residue pair energies are the reason for the complexity that necessitates methods such as Integer Programming and Tree Decomposition based threading. If it were not for these energies, local to global combinatorial optimization would work and thus dynamic programming would be a viable algorithmic choice for performing the alignments. The algorithmic implications of these energies are explored in more detail in Chapter 5.

The Twobody Cutoff method is derived from statistics representing the occurrence of residues pairs within a given radius.

$$E_{cutoff}(i_1, i_2, j_1, j_2) = \begin{cases} P_{score}(i_1, i_2), & \text{if } dist(j_1, j_2)) < 7.2\mathring{A} \\ 0, & \text{if } dist(j_1, j_2)) \geq 7.2\mathring{A} \end{cases} \tag{2.6}$$

$$P_{score} = \sum_{a \in A} \left(aaprob(i_1, a) \sum_{b \in A} (aaprob(i_2, b) P(aatype(i_1), aatype(i_2))) \right) \tag{2.7}$$

$$P(a, b) = \frac{Pair(a, b)}{N(a)N(b)} \tag{2.8}$$

In this formulation i_1 and i_2 denote the first and second target amino acid positions, respectively. Likewise j_1 and j_2 denote the first and second template residues

positions, respectively. $dist(j_1, j_2)$ represents the physical distance between the j_1 and j_2 residues in the template structure as measured from Cα to Cα. $Pair(a, b)$ is the number of times the amino acid types a and b where within the cutoff distance, while $N(a)$ and $N(b)$ are the independent frequencies of occurrences, as measured in the protein structure database. In the case of glycine, a hypothetical C_α is used.

DFire Energy

Dfire energy is a distance dependent energy residue pair energy function proposed by Zhang et al [Zhang et al., 2004b] [Zhang et al., 2004a]. Dfire is a statistically based distance-dependent pair-wise energy that has been set up into several distance dependent bins. Eqn 2.9 describes how the energy fits into the described energy formulation, and Eqn 2.10 describes the actual Dfire Energy.

$$E_{dfire}(i_1, i_2, j_1, j_2) = \bar{u}(aatype(i_1), aatype(i_2), dist(j_1, j_2)) \tag{2.9}$$

$$\bar{u}(i, j, r) = \begin{cases} -\eta RT \log \frac{N_{obs}(i,j,r)}{(r/r_{cut})^{\alpha}(\triangle r/\triangle r_{cut})N_{obs}(i,j,r_{cut})}, & \text{if } r < r_{cut} \\ 0 & \text{if } r \geq r_{cut} \end{cases} \tag{2.10}$$

where $\eta(= 0.0157)$ is a scaling constant, R is the gas constant, $T = (300K)$ is the temperature, $\alpha(= 1.61)$ is a tuning co-efficient, $N_{obs}(i, j, r)$ is the number of (i, j) pairs observed in the database for a given distance shell r. $r_{cut}(= 14.5\mathring{A})$ is the maximum cutoff distance, and $r\triangle$ is the bin width.

Gap Penalties

In addition to the regular energies, we have also included a gap penalty. It consists of two components. First, there is a standard affine gap penalty, with an opening penalty of 10.6 and an extension penalty of 0.6. The idea of gap energies will be further explored in Chapter 3. Those energies will be referred to as $E_{gapopen}$ and $E_{gapextend}$.

Total Equation

The E_{tot} that describes the total energy of the sequence structure alignment can now be described as the dot product of the weight array W and the energy function array E. We call the full set of energy function F:

$$F = [n, s, ss, cutoff, dfire, gapopen, gapextend] \tag{2.11}$$

and the total objective function is

$$E_{tot} = \min \sum_{f \in F} E_f W_f \tag{2.12}$$

2.2.1 Violations

A violation is an instance where the good alignment pair is ranked lower than a bad alignment pair. Violations are categorized in sets. Each individual violation provides little information, but as a set, an accurate view of the energy landscape can

be reconstructed. In the protein threading problem there are two possible sources of violations, fold recognition violations and alignment accuracy violations. For this study, we have concentrated on fold recognition violations as more relevant search points can be generated given one set of sample points.

Fold level pairs are defined in four different levels by set FL which has been generated by protein fold classifications defined by SCOP.

$$FL_{target,template} \begin{cases} 3 & \text{Same family} \\ 2 & \text{Same super-family} \\ 1 & \text{Same fold} \\ 0 & \text{No relationship} \end{cases} \tag{2.13}$$

Violations can be scanned by looking at every pair inside the same query and applying the criteria in Eqn 2.14. These inequalities hold true when the minimal energy is preferable and a maximal fold level is preferable.

$$(A, B) \in VIOL \text{ iff } W_{cur}E_{Q,A} < W_{cur}E_{Q,B} \text{ and } FL_{Q,A} < FL_{Q,B} \tag{2.14}$$

The number of violations can be viewed as an approximate measure of the distinguishing power of the weights set. More violations signals a declining ability to distinguish correct sequence structure alignments from the incorrect ones. Given that criteria, we can sample the points listed in Table 2.1. The first step is to

Table 2.1: Sample energy patterns that can be used for Violated Inequality Minimization

Sequence	Structure	Mutation	Singleton	Secondary Structure	Gap Open	Gap Extend	Dfire	Fold Level
d1foea2	d1foea2	-26721.7	-0.035982	-53.039	0	0	-43.5986	3
d1foea2	d1ki1b2	74.064	-4.09137	-40.054	18	110	-18.3928	3
d1foea2	d1v5pa_	2434.65	-1.41425	-39.458	13	52	-13.16	3
d1foea2	d1q67a_	2236.46	-1.03245	-40.856	16	322	-22.6869	2
d1foea2	d4ubpb_	-66.4022	2.23072	-29.802	21	253	-8.18113	0

Table 2.2: Row Differences for energy vectors in 2.1

Sample A	Sample B	Mutation	Singleton	Secondary Structure	Gap Open	Gap Extend	Dfire
d1foea2,d4ubpb_	d1foea2,d1q67a_	-2302.8622	3.26272	11.054	5	-69	14.50577
d1foea2,d4ubpb_	d1foea2,d1v5pa_	-2501.0522	3.64497	9.656	8	201	4.97887

multiply each of the energy vectors by the current weight set. Violations are then found using the product matrix and the criteria in equation 2.14, with every energy sample compared to every other sample. If a violation is found, a difference vector is created by subtracting the vectors, $A - B$. Some sample difference vectors are shown in table 2.2 for a weight vector of all ones.

It is possible to write the criteria for $VIOL$ such that that (B, A) would also belong to the set, but it is important to maintain sign correctness of the resultant difference matrix. Because of this property, positive elements in the matrix, represented in table 2.2, represent the occasions where a particular energy function was unable to distinguish the proper alignment.

2.2.2 Violation Minimization

The violations provided by statement 2.14 are viewed as a set

$$(E_{good}, E_{bad}) \in VIOL \tag{2.15}$$

For each of the pairs we examine the inequality

$$\sum_{f \in F} W_f E_{good,f} < \sum_{f \in F} W_f E_{bad,f} \tag{2.16}$$

This inequality has already been proven to be incorrect by statement 2.14, but this should not be the case given that the sum of a set of accurate energy functions should be lower for good alignments than for bad ones. These inequalities can be made to be true by the addition of 'slack' variables s.

$$\sum_{f \in F} W_f E_{good,f} < s + \sum_{f \in F} W_f E_{bad,f} \tag{2.17}$$

For every violation there is a slack variable s. It would be easy to make all violations true by setting all the s to some arbitrarily large value, but we wish to minimize the amount by which we have to 'tweak' the answer. It is in this formulation that we tune the value of the W set in order to minimize the total values of s. The minimal slack required will be the one needed to set the equation

$$\sum_{f \in F} W_f E_{good,f} = s + \sum_{f \in F} W_f E_{bad,f} \tag{2.18}$$

$$\sum_{f \in F} W_f (E_{good,f} - E_{bad,f}) = s \tag{2.19}$$

This would be analogous to changing the response of the function from being completely incorrect or simply being unable to determine. Examples of the terms $E_{good,f} - E_{bad,f}$ are shown in table 2.2. We wish to find the minimal set of $s \in S$,

where S is the set of slack variables for a given set of violations, manipulating the set W. These manipulations are governed by the constraint that

$$\sum_{f \in F} W_f(E_{good,f} - E_{bad,f}) - s = 0 \tag{2.20}$$

The total equation to manipulate is given as

$$\min_{W} \left[0 \times W + \sum_{s \in S}(s) \right] \tag{2.21}$$

We multiply W times zero, because while it is part of the minimization equation, we are not trying to minimize the values of W.

This method seeks to minimize $E_{good,tot} - E_{bad,tot}$, by adjusting the set of W. But $E_{opt} = min \sum W_i E_i$, which means that E_{opt} is dependent upon W, and thus once W has been adjusted, a different alignment will be chosen. This means that the method must be cyclical, moving and readjusting as $E_{opt,tot}$ adjusts in response to W. It also means that move must be 'slowed', to keep dramatic changes that seem good from one particular sampling from dominating the equations. This can be done by averaging W_{old} and W_{new}.

We have found that this optimization method works by iteratively adjusting weights and not immediately resampling E_{opt}. Rather at each iteration, a new E_{tot} is calculated based on the new set of W and the original set of E.

Table 2.3: Top 1,5 scores over the course of weight training

	Family		SuperFamily		Fold	
Round	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
1	46.18	64.23	16.99	28.95	3.30	15.38
2	80.47	89.14	40.87	64.44	27.94	51.51
3	83.49	90.13	42.6	67.55	32.71	52.50
4	84.92	90.33	44.34	66.04	34.23	55.41
5	83.78	90.94	42.87	66.46	33.71	56.2
6	83.84	90.05	43.72	65.97	32.54	54.08

2.3 Results

In order to study the behavior of VIM training, we have applied it to the training set described in Section 3.2. In this training method, proteins are threaded using the current weight set, and then 50 cycles of VIM training based on those threading results. After the 50 cycles, the proteins are re-threaded.

The overall Top 1 and Top 5 scores can be seen for the training set in Table 2.3. There seems to be a continuing improvements at different levels of the fold recognition abilities.

The behavior patterns of the same training regiment can be seen in Figure 2.1. We can see that the number of violations very quickly diminishes over training time. The weight parameters tend to jump during resampling, but continue to adjust during the static portion of the training.

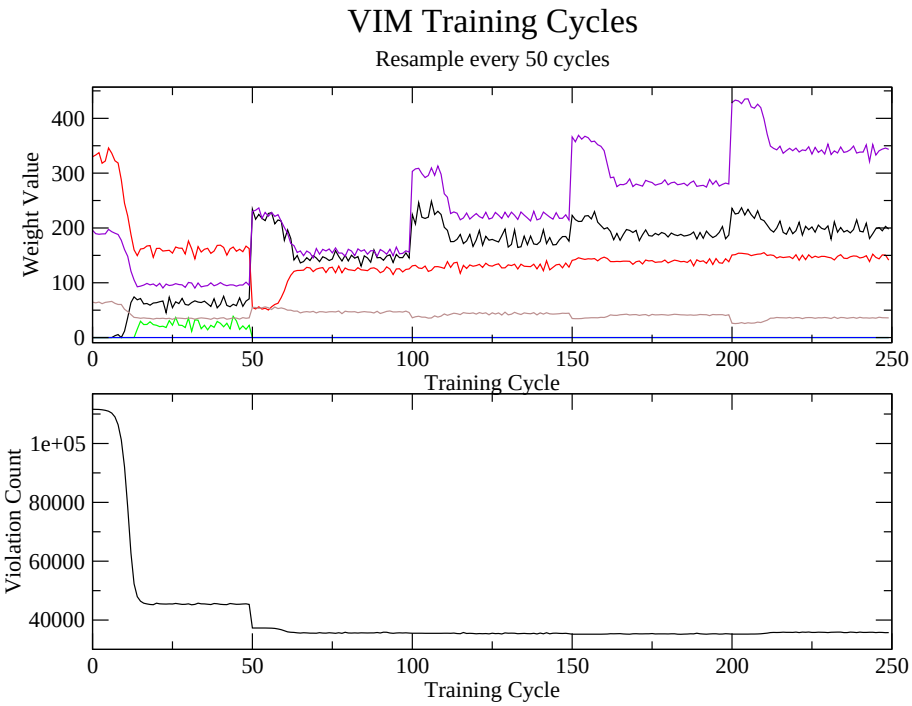

Figure 2.1: Violation count over time using Static VIM training

2.4 Discussion

It is tempting to link the weight parameter of a particular energy with it's approximate importance. One could point out that energies that have a poor ability to distinguish correct protein folds provide less noise to the total alignment when the weight parameters tend toward zero. It must be remembered, however, that weight parameters also serve the purpose of scaling the energy functions. Mutation energy is based on the log score of the probability divided by the expected probability. That means that the mutation energy scale is in the hundreds. This numeric behavior can be compared to the secondary structure matching probability, which described with the first decimal position. A set of weights set all to one would be dominated by the mutation energy because its values work in the highest order of magnitude.

2.5 Conclusions

These energies have been tested in multiple CASP competitions and have been integrated by several different threading protocols. After extensive testing with diverse protein libraries, these energies have shown a remarkable ability to distinguish correct protein sequence structure alignments from the incorrect ones. These energies also provide a vast amount of information for fold recognition. We have also shown that the Violated Inequality Minimization technique for weight parameter

optimization is well suited for quickly adjusting parameters to near optimal levels. We believe this set of energies and the methods used to optimize their weight parameters provide a solid basis for a protein threading platform.

Chapter 3

Variable Deletion Energies

Some of the text below has previously been submitted as:

Ellrott, K. Guo, J-T. Olman, V. and Xu, Y. "A Generalized Threading Model using Integer Programming with Secondary Structure Element Deletion", Genome Informatics, 17(2):248-258, (2006)

Ellrott, K. Guo, J-T. Olman, V. and Xu, Y. "Improving the performance of protein threading using insertion/deletion frequency arrays" (Submitted 2007)

3.1 Introduction

Protein threading, a technique for sequence/structure alignment, has played a key role in predicting protein structures in the past decade. The protein threading model is broken into two different but connected steps; protein sequence/structure

alignment and fold recognition. Protein sequence/structure alignment deals with how accurately an algorithm is able to map the amino acids from a target protein onto the structure of the protein template. Fold recognition deals with the ability to correctly identify if the target protein has been aligned to a native-like structural fold.

Most of the details in a threading model focus on how well an amino acid from a target sequence is aligned to a particular residue position on a known protein structure. For example, the energy functions used in our threading program PROSPECT include mutation, singleton, secondary structure match, and two-body interaction energies [Kim et al., 2003], which primarily concentrate on the positive space of the alignment, i.e. rewarding amino acid alignment. Deletion penalties, on the other hand, are a set of terms that describe how to penalize an alignment when gaps are introduced. There are two primary changes during protein evolution: mutation and insertion/deletion. A mutation event is the result of changing one amino acid to another and is evaluated by mutation energy matrices, such as PAM [Dayhoff et al., 1978] and Blosum [Henikoff and Henikoff, 1992].

Another event in protein evolution is the insertion and deletion of amino acids. These evolutionary changes are evaluated with gap penalty models in protein threading algorithms. While several gap penalty models have been proposed, the most widely used model for gap penalty is the simple affine model [Reich et al., 1984]. In this model there is a large penalty for opening a gap, or starting a deletion, and a smaller constant penalty for continuing that insertion/deletion. This can be viewed

as a simple linear function, $G = W_{open} + W_{const} * len$, where len is the length of the gap, W_{open} is the penalty for opening a gap, and every residue deleted is penalized by a constant penalty W_{const}.

This simple linear function can be easily implemented in a dynamic programming based alignment program such as the Smith-Waterman method, with a running time of $O(NM)$ where N is the length of the target sequence, and M is the length of the structural template. In addition to this linear penalty model, there are more sophisticated methods that have been developed. These methods typically attempt to formulate the penalty as non-linear functions [Qian and Goldstein, 2001] [Goonesekere and Lee, 2004] or use monotonic functions to avoid over-penalizing large gaps [Mott., 1999]. However, these non-linear gap functions cannot be optimized using traditional Smith-Waterman algorithm and require more advanced algorithms for sequence-structure alignment optimization [Dewey., 2001] [Madhusudhan et al., 2006].

Nonetheless, it is possible to use a nonlinear gap function, within the framework of the Smith-Waterman algorithm, if the function is monotonic and only dependent on local sequence/structure alignments. The penalty of a gap can be dependent on the probability of an amino acid being deleted or being inserted. Given these conditions, a set of local optimal decisions can still be aggregated to achieve the global optimality. Therefore, dynamic programming can still be used.

The patterns of insertions and deletions can be studied in a way that is similar to the generation of the Position Specific Score Matrices(PSSM) [Altschul et al.,

1997]. PSSMs have had a significant impact on secondary structure prediction and protein fold recognition [Yona and Levitt, 2002]. A PSSM is generated by finding homologous sequences in a non-redundant (NR) sequence database and aligning those sequences. The amino acid mutation patterns are used to create residue specific replacement scores. Using statistical analysis of alignments from a 'PsiBlast' search against the NR database, we can construct penalty functions that are based on insertion/deletion patterns specific to a protein family and the different portions of the sequence. These scores are not simply based on a global constant. For every residue, the percentage of times that it is deleted, or allows for an insertion, can be measured against a database of known sequences. We call this information the Indel Frequency Arrays (IFA). It should be pointed out that this type of energy, unlike some of other previously mentioned gap models, is only dependent on local sequence alignment and thus can be run in the same computational time as the Smith-Waterman algorithm. There has been similar position specific gap penalties suggested previously, such as the work by Lesk at al. [Lesk et al., 1986]. However, their work was based on different scoring values for differently assigned secondary structure values, and was not specific to protein families.

By using these IFAs in protein threading that we have developed, we have shown a noticeable improvement in the areas of fold recognition and alignment accuracy.

3.2 Methods

Alignment Strategy

We use dynamic programming version of our threading program, PROSPECT, for the study of these new deletion methods. PROSPECT has facilities for more advanced threading techniques that are capable of optimizing alignments utilizing residue pair energy information. However, we treat the Smith-Waterman algorithm as the greatest common denominator, because if the energy can be successfully applied to this method, then it will likely be successfully applied to a more complex algorithmic technique.

In our example of this method the optimal alignment is calculated by finding an alignment A with the total alignment score E_{tot} defined as:

$$\begin{aligned} E_{tot} = \min_{A}(&W_{mut}E_{mut}(A)+ \\ &W_{singleton}E_{singleton}(A)+ \\ &W_{secstruct}E_{secstruct}(A)+ \\ &W_{open}E_{open}(A)+ \\ &W_{const}E_{const}(A)) \end{aligned} \tag{3.1}$$

Where E_{mut} is the mutation energy, $E_{singleton}$ is the Singleton energy, $E_{secstruct}$ is secondary structure match energy; E_{open} and E_{const} represent the two aspects of

the affine gap function, the gap opening penalty, and the constant deletion for each residue removed; the set of Ws represent the weight parameters.

Optimal alignments can be found using dynamic programming by finding iterative solution of the values for $S_{i,j}$, with i and j both going from zero to the length of the target (l) and template (m) respectively. $S_{i,j}$ represents the alignment of the target residues from 0 to i and the template residues from 0 to j. Thus the total alignment is expressed as $E_{tot} = S_{l,m}$. The value of $S_{l,m}$ can be iteratively calculated with the formula S where $S_{0,0} = 0$:

$$S_{i,j} = min \begin{cases} S_{i-1,j-1} + E_{i,j} & \text{Match} \\ S_{i-1,j} + INS(i,j) & \text{Insertion} \\ S_{i,j-1} + DEL(i,j) & \text{Deletion} \end{cases} \tag{3.2}$$

If one selects an insertion then one sets $(i,j) \in I$. If you select a deletion then you set $(i,j) \in D$. Where I is the set of insertion operations, and D is the set of deletion operations. These equations refer to operations on the template, i.e. an insertion operation is an insertion on the template.

The energy $E_{i,j}$ for aligning a target position i to a template position j is defined as:

$$\begin{aligned} E_{i,j} =& W_{mut}E_{mut}(i,j)+ \\ & W_{singleton}E_{singleton}(i,j)+ \\ & W_{secstruct}E_{secstruct}(i,j) \end{aligned} \tag{3.3}$$

In the original model, the INS and DEL values where calculated as such:

$$INS(i,j) = \begin{cases} \text{If } (i-1,j) \in I & W_{const}E_{const} \\ \text{If } (i-1,j) \notin I & W_{open}E_{open}+ \\ & W_{const}E_{const} \end{cases} \tag{3.4}$$

$$DEL(i,j) = \begin{cases} \text{If } (i,j-1) \in D & W_{const}E_{const} \\ \text{If } (i,j-1) \notin D & W_{open}E_{open}+ \\ & W_{const}E_{const} \end{cases} \tag{3.5}$$

New Gap Energy Model

Deletion is the inverse operation of insertion. A deletion in the target is equivalent to an insertion to the template, and visa-versa. However, how these operations are

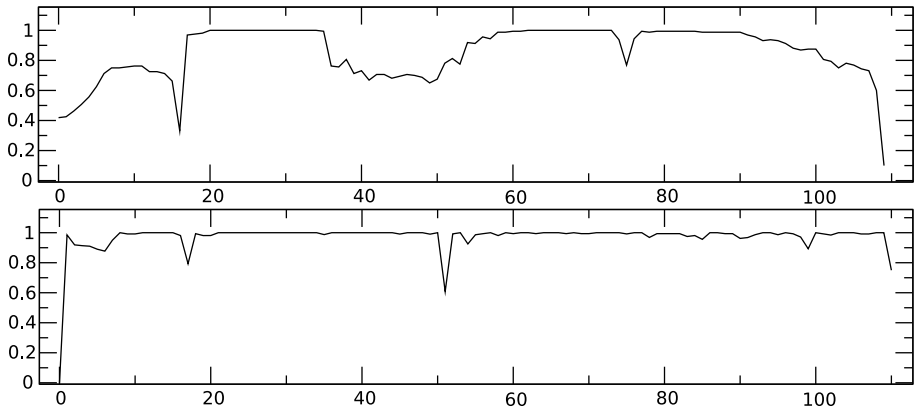

Figure 3.1: IFA information associated with the SCOP identifier 'd1qhoa2'. The top graph is the IFA information for deleting any of the residues. The bottom graph represents the IFA information for having an insertion before a given residue. A value of zero means that the indel can occur without penalty.

handled is different. The deletion occurs at a specific point, while an insertion occurs in between two residues.

Our study has shown that deletion and insertion probabilities are not equally distributed across the entire sequence and are unlikely to be similar for different protein sequences. A sample distribution can be seen in Figure 3.1. The probability of deleting a residue is not directly related to the probability of inserting a residue immediately before or after it, suggesting a simple deletion operation could actually encompass four different energies, insertion and deletion for both sequences being aligned.

As a result, the penalty feature previously described as E_{const}, a constant penalty for every insertion/deletion, can be replaced with a set of four features: E^q_{ins}, E^q_{del}, E^t_{ins}, and E^t_{del}, where t stands for template, and q stands for query or target.

In order to maintain the W coefficients each of the new penalty values needs to be scaled between 0 and 1, and multiplied by W_{const}.

Under this new model we redefine the $INS(i,j)$ and $DEL(i,j)$ functions as 3.6 and 3.7.

$$INS(i,j) = \begin{cases} \text{If } (i-1,j) \in I & W_{const}\left(E^t_{ins}(i-1) + E^q_{del}(j)\right) \\ \text{If } (i-1,j) \notin I & W_{open}E_{open} + W_{const}\left(E^t_{ins}(i-1) + E^q_{del}(j)\right) \end{cases} \tag{3.6}$$

$$DEL(i,j) = \begin{cases} \text{If } (i,j-1) \in D & W_{const}(E^t_{del}(i) + E^q_{ins}(j-1)) \\ \text{If } (i,j-1) \notin D & W_{open}E_{open} + W_{const}(E^t_{del}(i) + E^q_{ins}(j-1)) \end{cases} \tag{3.7}$$

Calculating Indel Profiles

The insertion/deletion profiles used to create the IFA are derived from alignments using Psi-Blast against the NR (non-redundant) protein database. Deletion energies are determined by observing the number of times a residue is deleted, or a residue being inserted before it.

The variable deletion functions are derived from the percentage of times that a particular residue is deleted across proteins in the same family, or allows an insertion in front of itself when being aligned to homologues found in the sequence database.

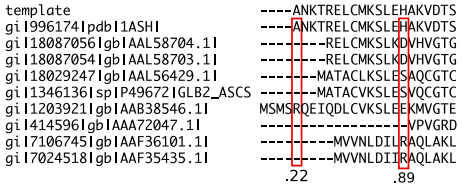

Figure 3.2: A sample multiple sequence alignment used to calculate B_{del}.

These arrays of percentages are referred to as B_{ins} and B_{del}. They are then formulated as energy functions such that $E_{ins}(i) = 1 - B_{ins}(i)$ and $E_{del}(i) = 1 - B_{del}(i)$. The beginnings of the B_{del} array can be seen in Figure 3.2. The top line represents the query sequence, and the lines below it represent the aligned homologues. The two highlighted columns show the percentage of deletions for those particular residues.

To make the calculations of the B_{ins} and B_{del} easier, we translate the standard two line text alignment returned back by Psi-Blast into a number array as shown in Figure 3.3. In this format array a represents the indices of the aligned subject sequence for each amino acid in the target sequence. If $a[i] = j$ then the ith amino acid in the target sequence is aligned to the jth amino acid in the template. Positions that are not aligned to any residue are represented by -1. After the above conversion, finding the insertions/deletions becomes a matter of referring to one array, rather than parsing two text arrays. To find the deletions, one simply scans the a array looking for the -1 entries, which represent the non-aligned residues. To find the insertions, one looks at all the non -1 entries in the array. For the ith

```
Target:-ABC--DEFGI----
Template:AABCAA--FGIZZZZ
TargetAlign:2,3,4,-1,-1,7,8,9
TemplateAlign:-1,1,2,3,-1,-1,6,7,8,-1,-1,-1,-1
```

Figure 3.3: The conversion from the sequence based model used to represent alignments, typically as outputted by Blast.
The bottom shows the same alignment in an easier hash table format. Each position represents the number of the position of the aligned residue in the opposite sequence. Deletions are represented as a -1.

residue that is followed by the next non -1 residue j, if $a[i] + 1 \neq a[j]$ then there is a gap. If the first non -1 entry is not 1, there is a pre-sequence insertion. Similarly, if the last non -1 entry is not aligned to the last residue of the subject, there is a post-sequence insertion. This information is then summed for each individual residue position and divided by the number of aligned sequences. This provides the percentage of times a residue is deleted or allows an insertion.

Training and Testing

We use two methods to evaluate the improvements of alignment accuracy the IFA method provides. First, we compared the alignment results with the output from FAST [Zhu and Weng, 2005], a structural comparison program. We chose FAST because of its efficient and accurate performance. FAST can correctly align 96% of the residue pairs in aligned regions of the 1033 protein alignments in the HOMSTRAD database [Mizuguchi et al., 1998] [Zhu and Weng, 2005]. As a common practice alignment is considered to be correct if the residue was aligned within 4 residues of the FAST-based structure-structure alignment position [Xu, 2005]. The

reported percentage accuracy is the percentage of residues placed within 4 residues of the correct position out of the total possible residue placements. The next method of evaluation is the MAMMOTH program [Ortiz et al., 2002]. MAMMOTH determines the statistical significance of the backbone structure created by predictive tools against the actual backbone structure of the target. We report the $-ln(E)$ score, for which a value greater than 4 is statistically significant.

Our training set is comprised of 300 SCOP [Murzin et al., 1995] domain entries from the ASTRAL 25 list [Chandonia et al., 2004], in which no entries would have higher than 25% sequence identity. Based on the results from FAST, the average sequence identity for the aligned pairs of this data set is 9.5%. Using the SCOP identifiers, we then compiled a list of all proteins that occurred in the same fold, super families and families. To find the optimal set of weights for each of the gap penalty permutation, we have applied 10 iterations of the Violated Inequality Minimization (VIM) [Zien et al., 2000] method for optimization to the entire set of weights. The training set is used to find a set of optimal weights for our original threading approach which uses traditional affine gap penalty. The same set of weights was used in the variable deletion model.

For the testing we used a set comprised of 724 proteins also derived from ASTRAL 25 that did not overlap with the original training set. We used the same SCOP table to determine relationship. For alignment accuracy analysis, sequence structure pairs were filtered by the FAST SN score. The SN score determines significance of the structural alignment created by FAST. Pairs with scores lower

then 2 were removed so that bad alignments would not create noise when analyzing the performance of threading results against the structural alignments. This left a testing set comprised of 3058 pair relationships, including 344 pairs in the same family, 1265 pairs in the same super family and 1449 pairs in the same fold level.

Fold Recognition by ZScores

One method for recognizing if the sequence and structure that have been aligned by threading are in the same fold family is to analyze the statistical significance of that alignment. This can be done by measuring the ZScore [Bryant and Altschul, 1995] of the alignment by comparing it to an ensemble of decoys with similar properties. Typically these decoy sequences are created by shuffling the original sequence. This randomly shuffled sequence is then threaded against the template using the same alignment procedure and the E_{tot} is calculated using the same set of energy functions. The mean μ and the standard deviation σ are calculated from samples of E_{tot} from the decoy sequences. The ZScore Z is calculated with the equation

$$Z = \frac{E_{tot} - \mu}{\sigma} \tag{3.8}$$

Fold Recognition by Gradient Boosting

Beyond using Zscore analysis there has been much research in the field of applying machine learning techniques to the fold recognition problem. Previous research includes techniques such as neural networks [Xu et al., 2002], SVMs [Xu, 2005],

and gradient boosting [Jiao et al., 2006] based functions. For this paper we use the gradient boosting method to measure improvements in fold recognition that may come from the use of IFA information.

To train this recognition technique we use the number of correctly aligned residues for the alignment, as defined previously in the 'Testing and Training' section, as the response value. The input vector include: all of the energies scores (unweighted), the Zscore, the individual Zscore statistics for each of the energies, the number of aligned residues, the number of aligned residues that are identical, the number of residue contact pairs that have both members align, and the number of contacts that have one of the partners deleted. This vector of features is similar to the one described in the fold recognition paper by F. Jiao et al [Jiao et al., 2006]. This information is then used to create a linear equation using the mBoost package available for the R statistical package.

Statistical Analysis of improvements in Alignment Accuracy

We describe our statistical model related to comparison of two methods as an experiment with multinomial distribution having three possible outcomes: Method 1 (IFA method) is better than Method 2 (original method) (probability P_{+1}), the two Methods are equal in their power (probability P_0), and Method 2 is better than Method 1 (probability P_{-1}), $P_0 + P_{-1} + P_{+1} = 1$. Our goal is to check hypothesis $H_0 : P_{+1} = P_{-1}$ against the alternative $H_1 : P_{+1} > P_{-1}$. For hypothesis checking we use two

tests: Pearson χ^2 test and λ-likelihood ratio test. Let N be the number of comparisons, K_{+1} the number of times Method 1 worked better,K_{-1} the number of times Method 2 worked better, and K_0 be a number of cases when Methods worked equally. The value of the Pearson statistics is $\chi^2 = \frac{(K_{+1}/N - P_{+1})^2}{P_{+1}^2} + \frac{(K_{-1}/N - P_{-1})^2}{P_{-1}^2} + \frac{(K_0/N - P_0)^2}{P_0^2}$, and the value of log-likelihood test is $\lambda = (\frac{P_0}{K_0}N)^{K_0}(\frac{P_1}{K_1}N)^{K_1}(\frac{P_2}{K_2}N)^{K_2}$. If H_0 is true then $P_{+1} = P_{-1} = 0.5 * (1 - P_0)$, and replacing P_0 with its maximum likelihood estimator K_0/N, we get $\chi^2 = \frac{(K_{+1}/N - p)^2}{p^2} + \frac{(K_{-1}/N - p)^2}{p^2}$ and $-2 * log(\lambda) = K_1 log(\frac{P_1}{K_1}N) + K_2 log(\frac{P_2}{K_2}N)$. The asymptotic distribution of the χ^2 and $-2 * log(\lambda)$ in our case is a χ^2 distribution with one degree of freedom.

Fold Recognition Analysis

There are different ways to measure fold recognition performance. We will concentrate on two different methods for fold recognition analysis, Top N analysis and sensitivity/specificity curves.

A Top N analysis uses the predicted fold score to sort the list of templates. Each one of the templates is denoted by its fold level similarity with the target structure.

These are indicated with numeric values for each of the levels of similarity:

$$S_{target,template} \begin{cases} 3 & \text{Same family} \\ 2 & \text{Same super-family} \\ 1 & \text{Same fold} \\ 0 & \text{No relationship} \end{cases} \tag{3.9}$$

For each Top N test, three query levels (3,2,1) Q are tested. For each query level, relationships $S > Q$ are removed. Thus if we are checking super-family(2), we remove all family(3) templates. Then, for each target that there exists a relationship for the level being tested, a positive score is noted if at least one of the relationships is within the top N of the sorted list. For our purposes we record Top 1 and Top 5. In order to do statistical analysis, a random set of 500 training proteins are sampled and the Top 1&5 numbers are measured. This sampling is repeated 1000 times, and the averages are reported.

While Top N measures the ability for the fold recognition to sort template lists, it does not necessarily provide a measure of a fold recognition techniques ability to answer the question of whether or not a template alignment is from the same fold. A fold recognition score needs to be able to classify a protein sequence/structure alignment with the same cutoff despite differences in protein length and composition. To measure the fold recognition ability, we plot a sensitivity/specificity curve for a range cut-off values. For the testing set of 724 proteins, we take 200 random proteins

and sample the sensitivity and specificity for a range of Zscore cutoff values between 0 and 30, we repeat this sampling 100 times to derive an average performance. This analysis is done for each of the three levels of similarity.

3.3 Results

Improvement in Alignment Accuracy

The overall average improvement for alignment accuracy is shown at different alignment levels in Table 3.1. The more distant two proteins are evolutionarily, the bigger improvement of the IFA method. At the fold level, alignment accuracy increase from 42.6% to 46.2%, an improvement of 8.5%. Once two proteins are in the same family, the amount of improvement decreases to 3.3%. A similar trend is observed if we separate protein pairs by their percentage of aligned amino acids. The lower the identity, the larger improvement that the IFA model provides. This trend can also be seen in Figure 3.4. The top section of the figure represents binned averages across different levels of sequence identity. The bottom section represents a segmented linear regression.

A specific example for alignment improvement shown in Figure 3.5 demonstrates the potential benefits of this information source. Both proteins were classified as 'winged helix' DNA-binding domains. The original model over compensated for C-terminal deletions. This caused the second helix to be aligned to the location where the first helix should be aligned. This cascaded into a series of mid sequence

Table 3.1: A comparison using different gap function.
The increase in performance is relative to the value obtained by the original model.

	Fast			MAMMOTH			
SCOP Alignment	Original	IFA	Increase	Original	IFA	Increase	Set Size
Fold	42.6	46.2	8.5%	11.5	13.2	14.8%	1449
SuperFamily	55.3	57.1	3.3%	13.9	15.2	9.4%	1265
Family	70.6	71.4	1.1%	14.5	15.5	6.9%	344
	Fast			MAMMOTH			
Sequence Identity	Original	IFA	Increase	Original	IFA	Increase	Set Size
0% – 5%	36.2	38.3	5.8%	13.4	15.3	14.2%	909
5% – 10%	51.8	55.2	6.6%	12.5	14.0	12%	1527
10% – 15%	68.4	70.4	2.9%	12.6	13.5	7.1%	487
15% – 20%	76.4	80.1	4.8%	11.8	12.5	5.9%	107
20% – 100%	91.4	93.9	2.7%	13.4	13.7	2.2%	28

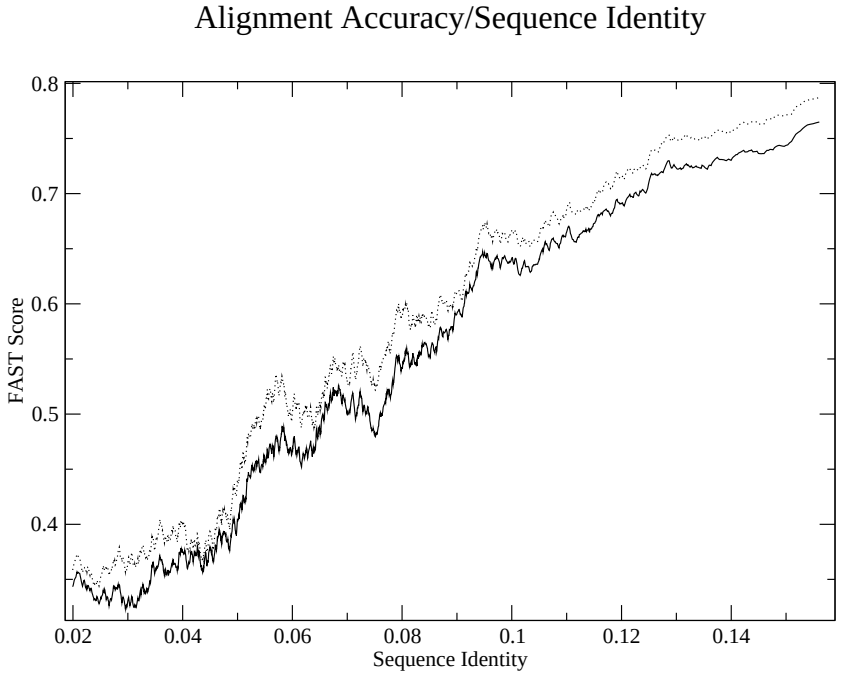

Figure 3.4: The running averages of the two methods.
The dotted line represents the IFA model.

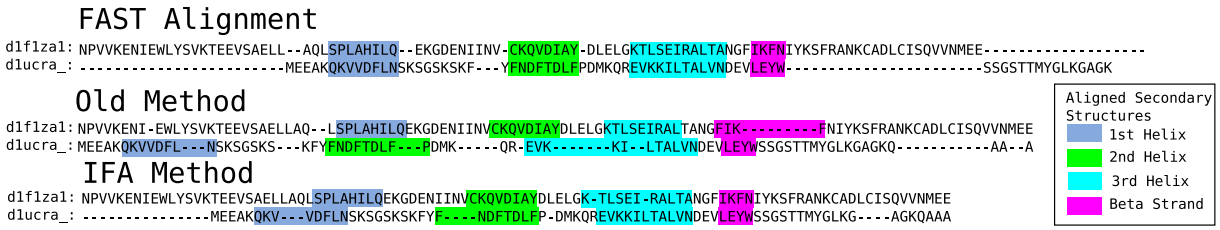

Figure 3.5: An example alignment between SCOP domains 'd1f1za1' and 'd1ucra_'. Each block represents an assigned secondary structure element.

deletions that smeared two helices together. Our model increases the alignment accuracy from 26.5% to 73%.

In order to show that these improvements in alignment accuracy were not the product of random statistical fluctuations we have also analyzed the comparative performance of the two methods on the same alignment pairs. We counted the number of times the new model has led to an improvement in alignment accuracy, shown in Table 3.2 . We have also calculated the statistical significance of these numbers. Using the statistical analysis described in the Methods section, we can calculate the p-value of the hypothesis that improvements are random. For the testing set with FAST based alignments, we get the values $K_{+1} = 1464$, $K_0 = 882$, and $K_{-1} = 712$, which for the first statistical testing method leads to the p-value=3.55×10^{-111}. This shows that the difference in performance of two methods can not be explained by pure chance, indicating the superiority of Method 1, our new IFA method.

Table 3.2: Side by side comparison of old method and IFA

Level	IFA	Original	Tie	Pearson χ^2	λ likelihood
FAST					
All	1464	712	882	3.55×10^{-111}	1.58×10^{-113}
Fold	667	362	420	5.00×10^{-38}	1.29×10^{-38}
SuperFamily	647	281	337	2.97×10^{-61}	5.60×10^{-63}
Family	150	69	125	3.02×10^{-12}	1.5×10^{-12}
MAMMOTH					
All	2383	282	393	0	0
Fold	1170	138	141	0	0
SuperFamily	973	118	174	6.72×10^{-289}	0
Family	240	26	78	2.94×10^{-73}	1.29×10^{-84}

Improvement in Zscore based Fold Recognition

We started fold recognition analysis using Zscore based analysis for Top 1 and Top 5 scoring. These results can be seen in Table 3.3

There is a noticeable improvement in sorting ability when the IFA information is added to the model. As with alignment accuracy, the most notable improvements occur for fold level alignment pairs. A similar improvement is seen in fold recognition when analyzing the sensitivity and specificity curves in Figures 3.6, 3.7, 3.8. The dotted line represents the sensitivity/specificity curve for the IFA model. In each of the the fold levels, the improvement is uniformly better. This means that for every one of the possible cutoffs in the old model there is a cutoff in the new model that provides a better specificity and sensitivity. As with the alignment accuracy, the most dramatic improvements come in the fold level recognition.

Table 3.3: Zscore based fold recognition results.

	Family		SuperFamily		Fold	
Threading Method	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
Original	67.4	79.3	47.9	62.6	18.6	31.8
IFA	69.8	82.3	50.3	63.3	21.4	38.5
Improvement	3.6%	3.8%	5.0%	1.1%	15.1%	22.0%

Table 3.4: Gradient Boosting based fold recognition results.

	Family		SuperFamily		Fold	
Threading Method	Top 1	Top 5	Top 1	Top 5	Top 1	Top 5
Original	66.7	80.7	48.6	68.4	22.1	46.1
IFA	70.7	83.3	53.6	68.3	25.8	49.7
Improvement	6.0%	3.2%	10.3%	-0.0%	16.7%	16.4%

Improvement in Gradient Boosting based Fold Recognition

Looking at Table 3.4 a similar pattern in improvement as the Zscore table appears. The most significant improvements are in the Fold level pairs. Comparing Gradient Boosting to the Zscore method, there is a great deal of improvement in the Fold and Super Family level pairs. Looking at the sensitivity/specificity curves, one can see the improvement in fold recognition available using gradient boosting techniques.

Improvement in Machine Learning based Fold Recognition

We have already show an improvement in fold recognition when the IFA model is applied to a gradient boosting based learning method. This improvement in fold recognition should carry over to the other machine learning based techniques that have been used previously, such as Neural Networks and SVMs. These methods are trained to predict the the number of correctly aligned residues. This is based on

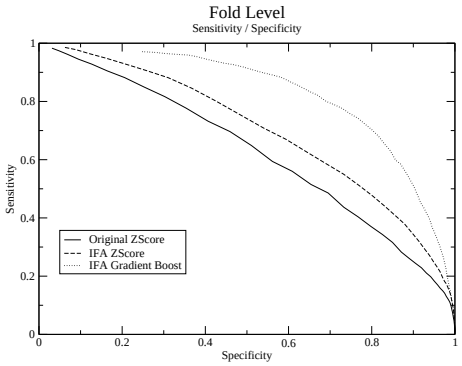

Figure 3.6: Sensitivity and Specificity for Fold Level pairs

the assumption that the closer two proteins are in terms of their fold families, the more amino acids that are likely to be correctly aligned. Therefore, the better the predicted alignment accuracy, the better the fold recognition method is likely to be. When training machine learning techniques previous research has used a vector to represent a set of features from an alignment, such as the values from the different energy terms, and trained them for fold recognition using the alignment accuracy as a measure. For these techniques, the more correlated a given feature is to the alignment accuracy, the easier it will be to train the regression function. Therefore, we can predict the benefit to fold recognition a feature will have by measuring its correlation with the alignment accuracy. We test our new energy functions by comparing the correlation coefficients of: E_{const}, E^t_{del}, E^q_{del}, E^t_{ins}, E^q_{ins} with the alignment accuracy.

We compared the correlation coefficients between the total sum of each of the energies and the accuracies of the alignment that produced that total energy. In

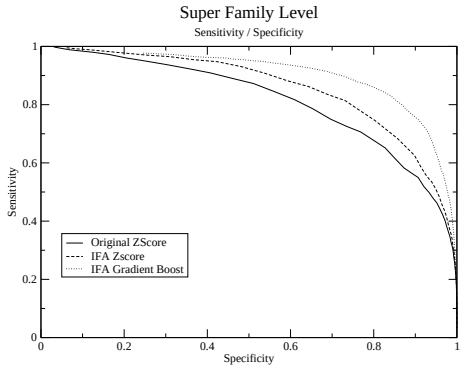

Figure 3.7: Sensitivity and Specificity for Super Family Level

Table 3.5 we have compared two different threading result sets. The first set is the alignments produced using dynamic programming and the constant deletion method, while the second set of results is produced using the IFA energy. The combined energy is $E_{var} = E^t_{del} + E^t_{ins} + E^q_{del} + E^q_{ins}$. The variable deletion energies, once combined, are very well correlated to alignment accuracy, indicating a good ability to distinguish correct alignments. The ability increases even more once they are used to optimize the alignment, as in the variable deletion threading set. First, the threading set comprised of alignments created with E_{const} and next, the set of alignments optimized using the set of variable gap penalties. As we can see in Table 3.5, individually each of the separate variable deletion penalties is not as correlated as E_{const}. However, once they are summed together, and used to optimize the alignment, their correlation increases greatly. This makes sense, because each of the variable deletion penalties is only 1/4th of the total deletion energy. The increase in correlation from -0.18, using the original model for alignment and fold recognition,

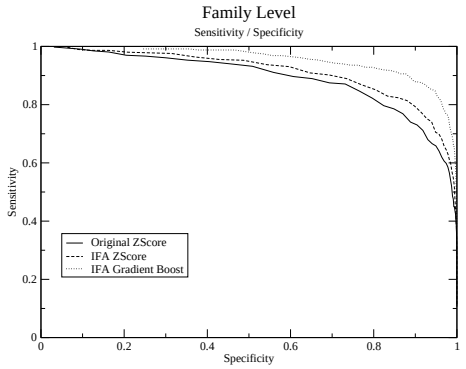

Figure 3.8: Sensitivity and Specificity for Family Level

to -0.27, for using the new model, should correspond to a greater ability to correctly differentiate correct fold from incorrect fold.

3.4 Discussion

We have demonstrated our new deletion model in the context of protein threading. This new energy seems to work best in the context of distantly-related threading models. The variable gap penalty by Madhusudhan et al [Madhusudhan et al., 2006] concentrated on the performance improvements of variable gap penalties for protein alignments with sequence identity spanning the range of 20-40%. Our profile-based variable deletion energy has its best improvements in the low homology range, from 2-10% sequence identity, the so called 'twilight zone' where both fold recognition and threading alignment accuracies are in desperate needs for improvements. As seen in Table 3.2, the lower the sequence identity, the more the IFA can improve the accuracy of sequence alignment. Not only did Indel Frequency Arrays improve

Table 3.5: The Correlation coefficients of the gap energies, as applied to the two threading method results.

Feature	Constant Deletion	IFA
E_{open}	-0.29	-0.29
E_{const}	-0.18	-0.11
E^t_{del}	-0.12	-0.15
E^t_{ins}	-0.10	-0.14
E^q_{del}	-0.11	-0.13
E^q_{ins}	-0.10	-0.16
E_{var}	-0.17	-0.27

the alignment accuracy, we also have shown that it is a statistically significant improvement. At higher sequence identity levels, where the variable deletion penalty starts to lose some of its advantage, it does not cause an increase in false positive. So it can be used safely, regardless of the level of homology. We have shown that our energy fits within the Smith-Waterman alignment framework, and it is also theoretically possible to incorporate it into the algorithmic methods suggested by Madhusudhan et al [Madhusudhan et al., 2006].

We have also shown that the application of IFA information during Zscore analysis yields noticeable improvements in fold recognition sorting and sensitivity. Again, these improvements were most apparent with 'twilight zone' protein pairs. We have also shown this performance increase when applied to the Gradient boosting based regression method, and also shown that this information should be usable by other machine learning techniques. In addition to the improved correlation with the response variable, these other machine learning techniques should benefit from increase

accuracy of the Zscore information. Very often machine learning based fold recognition methods will use the Zscore as an input feature, so any increase in it's accuracy will help. We also found an increased alignment accuracy correlation coefficient, which should translate into more accurate fold recognition.

Future work could explore the linking of secondary structure dependent deletion functions, such as one suggested by Lesk at al. [Lesk et al., 1986]. The two deletion functions are not mutually exclusive, and could be used simultaneously.

3.5 Conclusion

We have shown there is large amount of information inherent in the insertions and deletions during protein evolution. This information can be determined by analyzing sequence alignments with homologous sequences. Once applied, this technique can improve protein threading alignment accuracy. We have shown that this information can be applied to the Smith-Waterman sequence alignment algorithm without added complexity. These energies can also be added to more complex methods, such as integer programming [Xu et al., 2003] [Ellrott et al., 2006]. We have also shown that this information adds in the ability to correctly sort and identify correct fold pairs. When applying fold recognition techniques to genome level protein structure analysis, these sorts of improvement could help to identify fold families for hundreds of previously unidentified hypothetical proteins.

Chapter 4

Conserved Substructure Analysis for Threading Refinement

4.1 Introduction

As techniques in protein threading techniques have advanced, improvements in alignment accuracy and fold recognition have seen a diminishing return. Most common fold recognition programs work from a similar set of energies. A broad sample of these energies is provided in Chapter 2. Part of the problem is the limited amount of energy available for a given protein sequence. Most of the functions that relate to protein sequence/structure alignment relate to how a given residue in the template

structure will react to the placement of one of the amino acids from the target sequence. For a given protein sequence, database searches and heuristic analysis will provide approximate mutation matrices and secondary structure prediction. The secondary structure prediction is usually a neural network based assignment. These assignments are usually accurate 85% of the time. But the only input to the neural network is the mutation matrix that was originally produced by the Psi-Blast search. Any other information about the target amino acid sequence is hard to come by.

Threading works by scanning a library of non-redundant protein structure templates in an effort to determine which of them may be closely related. Typically these structure templates are compared individually and sorted by some type of scoring mechanism. It is known that proteins frequently have common substructures, even if their global structure is different. This means that even when scanning a non-redundant library certain sub-patterns will be repeated. But because each of these comparisons is done separately, this potential source of information is ignored. Given these principles we propose that even if the global alignment of a target sequence to a template structure is incorrect, it portions of the alignment may be locally correct for the common substructures.

We propose using the set of threading results of a single target sequence against the library of known proteins to provide a library of structural conformations for the different portions of the target sequence. We believe that it is important to look at the way that protein threading is done, and begin to analyze the results of threading against a library. Figure 4.1 shows an example of a target protein sequence that

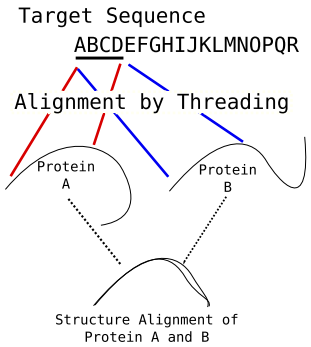

Figure 4.1: An example of a section of a target protein sequence that has been aligned to two similar protein sub structures

has been aligned to two separate protein fragments with similar structure. Even if a protein structure is not of the same fold as the target structure, there is a chance that sub components of the alignment are well aligned. This may provide us with new information that can be used to re-thread the protein, and provide better clues for fold recognition.

Our method for data gathering can be compared to Psi-Blast [Altschul et al., 1997] and how it differs from traditional Blast. Blast works by scanning the database once. Psi-Blast scans the library, then uses statistically significant hits to calculates a mutation profile for the target sequence and then re-scans the database. In our method, the preliminary threading search is used to create the pool of sequence structure alignments. From these structural fragments we seek to identify significant clusters of protein structure.

Protein structure is comprised of a tertiary structure built up from conserved sub structure motifs. There are methods used to determine possible protein structure

by assembling these fragments, however these methods are typically more efficient when there is no know homologue. If there is a related protein available in the known database, threading will provide a better structural template from which to build a model.

Traditionally threading has been concerned with providing proper alignments using sequence determined characteristics of the query protein. The point specific score matrix and the predicted secondary structure of the protein can be determined from the query sequence alone.

4.2 Methods

The technique starts with normal threading by dynamic programming. This technique has been outlined in previous papers [Kim et al., 2003] [Xu and Xu, 2000]. For preliminary threading optimization is done with a set of possible energies used to find the optimal alignment for a given

$$\begin{aligned} E_{tot} = min_t(&W_{mut}E_{mut}(A)+ \\ &W_{singleton}E_{singleton}(A)+ \\ &W_{secstruct}E_{secstruct}(A)+ \\ &W_{open}E_{open}(A)+ \\ &W_{const}E_{const}(A)) \end{aligned} \tag{4.1}$$

Where E_{mut} is the mutation energy, $E_{singleton}$ is the singleton energy, $E_{secstruct}$ is the secondary structure matching energy, and E_{open} and E_{const} which describe energy for opening and extending a gap. Further description of there energies can be found in the papers by Ellrott et al. [Ellrott et al., 2006] or Xu et al. [Xu et al., 2003] and in Chapter 2.

For this research we have proposed two methods for describing local protein structure patterns. The first is a distance matrix model, and the secondary is an array of phi-psi angles.

A protein fragment can be described as a matrix of distances between the different $C\alpha$ atoms along the backbone of the local structure.

$$D_{i,j} = dist(C\alpha_i, C\alpha_j) \tag{4.2}$$

We define the D matrix to be the size $l \times l$. For our set of experiments we have set $l = 9$, as it is a window size frequently used by other sub-structures programs [Simons et al., 1997, Rohl et al., 2004]. Because the description of the matrix is centered on the residue of interest, the window extend by 4 residues in either direction. For a window size of $l = 9$ we set the half window size $h = 4$. Giving these definitions, the distance between two matrices can be calculated as

$$MDist(Da_i, Db_j) = \sqrt{(\sum_{x=i-h}^{i+h} \sum_{y=j-h}^{j+h} (Da_{x,y} - Db_{x,y})^2))/(l^2)} \tag{4.3}$$

This can be modeled as a threading energy using the formulation

$$E_{matrix}(i,j) = -\exp^{1-MDist(Da_i,Da_j)} \tag{4.4}$$

In this format, the energy function starts at -1 for identical fragments, as the distance increases the score goes to 0 where is levels off. This is so that only information that 'agrees' with the prediction is used. Alignments that completely disagree with the prediction are not overly penalized. This is to counteract the possible effects of bad information.

Once these energies have been suggested, a new E_{tot} is proposed, with $W_{matrix}E_{matrix}(A)$ added to the formulations or $W_{phipsi}E_{phipsi}(A)$. These new E_{tot} formulations will be referred to as $E_{tot,matrix}$.

4.2.1 Fragment Selection

Each residue in the target sequence calculates its matrix independently. The profile matrix is selected from an averaging of selected representative fragments derived from threading results. Fragments are only included if they span the whole length of the window. This means that the first and last four residues of the target don't have profiles.

In order to train for the selection of proper fragments, the training protein set was used to collect samples of target sequences aligned to target structures using

threading. Then, the original protein structure backbone was aligned to the template backbone using the threading derived alignment, and the RMSD was calculated. The threading energy profile for that section is recorded, including mutation, singleton, secondary structure, target residue deletion energy, Dfire, and twobody cutoff energy. The gradient boosting technique [Jiao et al., 2006] is used to train a linear equation the energy profile against the RMSD of the fragment.

This trained linear regression is then used to predict RMSD values of the target fragments against the template protein when searching for fragments to use in the creation of the profile. There is a cutoff of a predicted 1.5 RMSD for sampled fragments, and only the top 10 scoring fragments are used for the matrix averaging.

4.2.2 Testing and Training

The testing and training method is comprised of the same protein sets and evaluation methods as described for the IFA training in section 3.2. In short the training set is comprised of 300 proteins, and a testing set of 724 proteins which comprises 3058 fold pairs. For evaluation there are two, FAST and MAMMOTH, used to score alignment accuracy.

In addition to the methods for measuring increase in alignment accuracy performance, we use a method to measure the accuracy of the structure descriptions that have been generated using this technique. For this test we use a subset of the original testing set of 100 proteins, and scan these against the standard default template library. The default template library is a representative set of proteins selected from

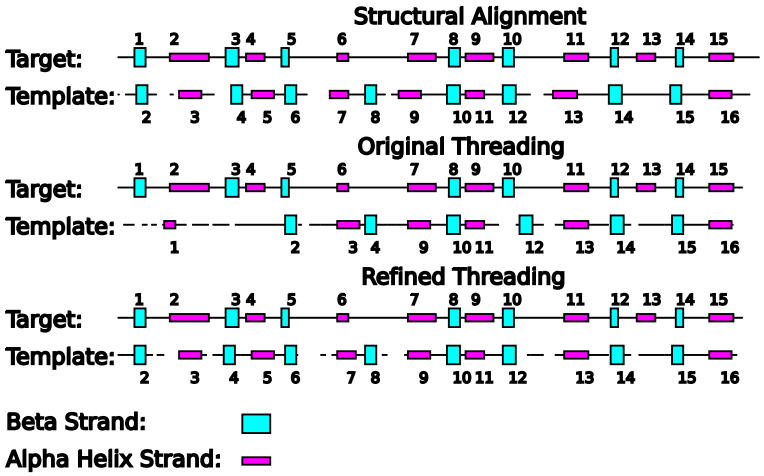

Figure 4.2: An example re-threading of the target protein d1hg3a_ to template d1vc4a_

the Protein Data Base (PDB) by a program called PISCES [Wang and Dunbrack, 2003]. For our particular experiments, the set comprises 4683 protein chains.

4.3 Results

A specific point example of an alignment improved by the addition of the conserved strucure energy can be seen in Figure 4.2. In this particular example, the refined threading accuratly places more of the template cores onto the target structure. In first pass of threading, the N terminal half of the protein is completely wrong. The addition of the conserved structure information helps to fix these problems.

Analysis of the alignment accuracy results for the entire testing set, shown in Table 4.1, seem to indicate a measured improvement in alignment accuracy as a result of the application of the protein structure refinement energy to threading.

Table 4.1: A comparison using different gap function.

	Fast			MAMMOTH			
SCOP Alignment	Original	IFA	Increase	Original	IFA	Increase	Set Size
Fold	38.38	41.37	7.8%	10.73	11.62	8.28%	1854
SuperFamily	52.13	53.25	2.15%	13.73	14.49	5.56%	1410
Family	69.66	70.64	1.41%	14.26	14.80	3.79%	352

The statistical significance of this improvement is detailed using methods originally described in section 3.2. The results from this analysis, shown in table 4.2, indicate that the improvement is more than just a random fluctuation.

4.4 Discussion

There seems to be a measured increase in the improvement of alignment accuracy. This is also a good indicator that there is a certain amount of accuracy in the profiled matrix data. How this information can be used in other applications, such as mini-threading, still needs to be studied.

Initial analysis indicates that the application of this energy provides negligible benefits for fold recognition. It is in contrast to the favorable results that where seem with the application variable deletions penalties in Chapter 3. This does not rule out the possibility using this information for fold recognition, as it may be applicable for more sophisticated recognition techniques.

Table 4.2: Side by side comparison of the Original Alignments and the Structure Refined Alignment

Level	IFA	Original	Tie	Pearson χ^2	λ likelihood
FAST					
All	672	1326	1619	2.30×10^{-91}	4.31×10^{-93}
Fold	321	673	861	9.19×10^{-53}	6.03×10^{-54}
SuperFamily	288	522	601	3.01×10^{-28}	1.15×10^{-28}
Family	63	131	159	1.11×10^{-09}	6.75×10^{-10}
MAMMOTH					
All	527	2324	766	0	0
Fold	278	1240	337	1.05×10^{-262}	4.56×10^{-284}
SuperFamily	204	880	327	3.49×10^{-181}	2.24×10^{-195}
Family	45	204	104	8.26×10^{-43}	2.09×10^{-46}

4.5 Conclusion

With this work we have shown a method to collect information from protein threading results. We have applied this method for both improvement in alignment accuracy and prediction of sub-structure features. In both tests, it has shown that the method provides a viable source of information that will enrich protein structure prediction.

Chapter 5

Integer Programming Based Threading

Some of the text below has previously been submitted as:

Ellrott, K. Guo, J-T. Olman, V. and Xu, Y. "A Generalized Threading Model using Integer Programming with Secondary Structure Element Deletion", Genome Informatics, 17(2):248-258, (2006)

Two of the energy functions touched on the Chapter 2, Twobody Cuttoff and DFire, are a different class of energy than a regular mutation or deletion energy. These energy are both dependent on the alignment of two residue pairs. As originally noted in the introduction chapter, when these types of energies are added to

the optimization problem, the complexity become NP-hard. But the use of these energies is desirable because structural characteristics will be more predominate when there is little to no sequence identity.

This chapter begins with the integer programming model for solving the protein threading that was previously described by other research and augments it to add additional features. By implementing a very minimal change in the overall structure of the problem, the model allows for alignments with deleted cores. Prior to this model, cores were static and undeleteable.

5.1 Introduction

Over the past decade there has been much work done on protein structure prediction by threading approach. The basic idea of protein threading is that by aligning the sequence of a protein with an unknown structure to a known structure one would gain a large amount of structural information [Bowie et al., 1991] [Jones et al., 1992]. This is done with a variety of statistically derived energy functions that score the placement of a residue onto a template structure. There are a number of techniques to optimize the alignment to find the best score, one of the most classical of which is dynamic programming. Dynamic programming is predicated upon the assumption that a global solution to the problem in question can be obtained by calculating a local solution and then expanding upon it. The energy functions used to judge an alignment describe the placement of a residue from the target onto a position in the

template protein. In simpler models this is the energy of placing only one residue onto one target position. However there is a certain amount of information that can be utilized by looking at multiple dependent residues that are simultaneously placed on different positions. The first level of this would be to look at placing two residues onto two target protein positions. This placement would then be judged by a residue pair energy function. By using this structural information, one could begin to align targets even when there is no sequence similarity with the target. However, with the addition of residue pair terms, the original assumption of a globally optimal alignment being able to be constructed from local optimal no longer holds true. This is because placing two residues at locally non-optimal positions may produce a pair energy low enough to produce a global optimal. These pairwise energies cause the algorithmic complexity of solving the problem to increase dramatically. The problem of optimizing of an alignment, given pairwise terms and allowing for gaps, has been shown to be NP-hard. [Lathrop, 1994].

Since the middle of 90s, a number of rigorous threading algorithms have been developed to take pair-wise energy into account. The first rigorous threading algorithm that considers pair-wise interactions was a branch and bound algorithm developed by Lathrop and Smith [Lathrop and Smith, 1996] though its actual computing time and the practical usefulness have not been well documented. Our group developed a threading program, PROSPECT, which solves rigorously the globally optimal threading problem using a divide-and-conquer strategy [Xu et al., 1998]. Its practical usefulness and the value in rigorously solving the threading problem were

demonstrated through the prediction server of the program [Y. Guo, 2004]. The threading problem was later formulated as a Linear Integer Programming (LIP) problem and was implemented as a computer program, RAPTOR [Xu et al., 2003]. The authors of RAPTOR took advantage of the extensive research results in the area of LIP to make the program run much faster than PROSPECT though the same set of energy function is used. It was convincingly demonstrated, through applications of programs such as PROSPECT and RAPTOR at the CASP contests, that threading programs with guaranteed global optimality do have an advantage over programs without this property [Fischer et al., 2003] [Sippl et al., 2001]. As discussed earlier, finding an optimal alignment that allows for gaps, while considering residue pair energy terms, has been shown to be NP hard. In order to make the problem tractable, certain concessions have to be made. First is the concept of the 'core'. A core is a secondary structure element involved in long-range interactions. The idea of core alignment is illustrated in Figure 5.1. These structural elements are recognized as not only being important for the basic structure of the protein, but are also typically conserved across homologues. On the other hand, changes in the loop region of the protein usually do not affect the overall structure of the protein.

In our study, we define a 'core' as a secondary structure element that is within range to have interactions with any other secondary structure element. For practical reasons, no gaps are allowed in cores and only in between these cores are the pairwise interactions actually considered. The loop regions can be aligned using simple dynamic programming method once the cores that occur immediately before

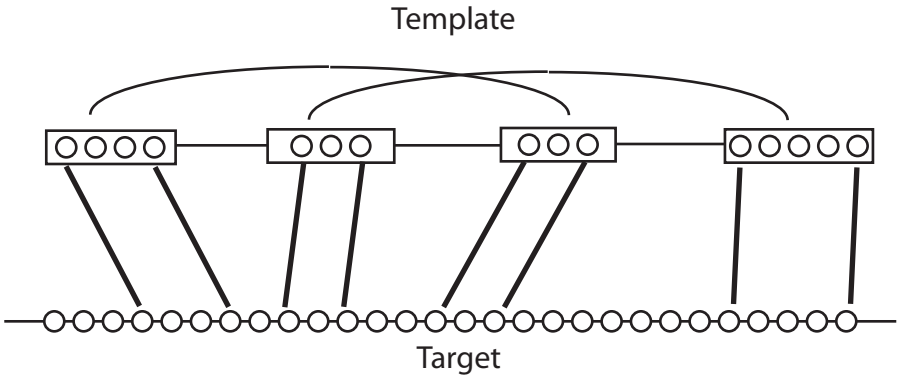

Figure 5.1: An example of template cores being aligned into a target sequence The energy of placing the first core also depends on the placement of all of the other cores that it is connected to.

and after the loops have been placed. With the concept of a core, the problem that was once NP-hard is now tractable, though still very difficult. Current integer programming threading approach has achieved good prediction accuracy. However, current integer programming forces every core of the template to be aligned to a position of the target sequence to reduce the computational complexity. For very distant homologues it is possible that some secondary structure elements have been deleted in one protein when compared to its structural homologs or analogs. A simple example of this is shown in Figure 5.2. Forcing cores into the alignment, when they should be deleted, can have a cascade effect of pushing other things out of optimal position to make way for the extra sequence.

In this study, we expanded the integer programming technique to allow for the deletion of interacting secondary structure elements, which was not possible in previous versions of the program. But before a core can be deleted, one must consider

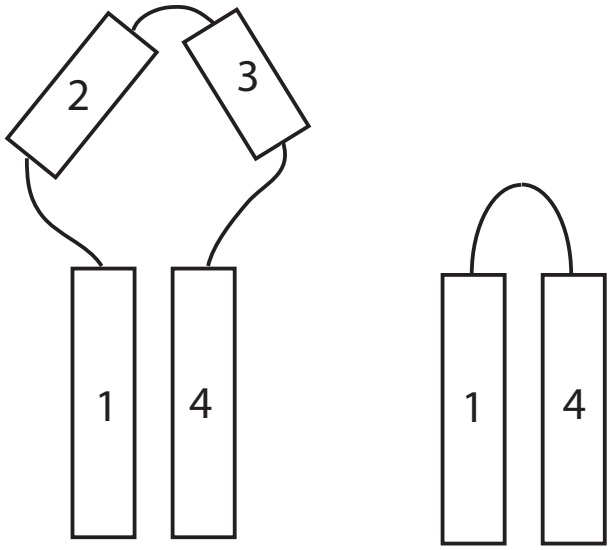

Figure 5.2: An example where cores 1 and 4 form a conserved substructure. In order to properly discover the alignment, one needs a model that allows for cores 2 and 3 to be deleted.

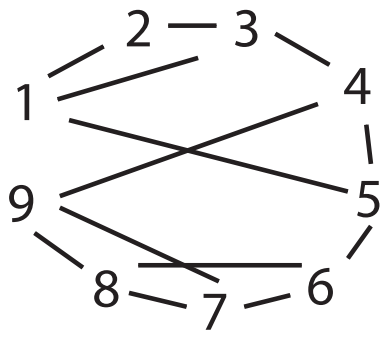

Figure 5.3: The connection graph shows which cores have active pairwise energies. Sequential cores are always connected. For unconnected cores, the pairwise energies are not considered while making the alignment.

the implications on the core connection graph. Figure 5.3 shows a connection graph for the cores in a protein. Cores that are not connected have no energy interactions that are considered in the optimization of the alignment.

Despite the fact the graph is not fully connected, the sequential order of the cores is preserved. This is possible because of a form of cascading logic. There exists a path of connections between every pair of cores, even if the cores are not directly linked. So while we may not care about the interaction between core 2 and core 4, we know that core 3 comes after core 2 and that core 4 comes after core 3. This implies that core 4 comes after core 2. However, if core 3 is deleted, this implied relationship between 2 and 4 is broken. In the connection graph there needs to be a path between every pair of core, otherwise the implied relationships break down. If any core can be arbitrarily deleted from the connection graph, the only way to make sure these paths are maintained is to make the graph fully connected. We will

show a method to allow for deletions that utilizes much less memory then the fully connected model would use. In order to achieve this, we need a way to allow core 2 to remain in the alignment, even though it has been deleted. To maintain the sequential alignment of the cores, we introduce the idea of an imaginary 'deletion state' that the core can enter into, while being aligned to the target sequence. This allows it to be deleted, but remain a component of the connection graph. This deletion state allows us to add core deletions to the model by increasing the number of residues of the target sequence, and not the number of interactions.

The amount of memory needed for an Integer Programming problem can be approximated by the size of the matrix needed to describe the problem. In protein threading model, the size of the objective function is proportional to $LN + L^2I$, where L is the length of the target, N is the number of cores, and I is the number of interactions. The number of cells in the matrix is proportional to $LN + LNI$.

At first glance, it would appear to be advantageous to minimize the size of L as the objective function is proportional to L2 and the cell count is proportional to L. Because allowing core deletions would break the connectivity of the connection graph, one could fix this by making the graph fully connected, causing the model's I to increase. An increase in I would only increase the size of the objective function by I. However it is important to point out that the I grows almost linearly with N, within the range of normal sized proteins (Figure 5.4). If the graph was to become fully connected then $I = N(N-1)/2$, thus the objective function would be proportional to $LN+L^2N^2$ and the cell count would grow proportional to $LN+LN^3$.

On the other hand, with the concept of imaginary core states the memory usage is equivalent to a problem with a target sequence of twice the length.

5.2 Methods

5.2.1 Integer Programming

In order to solve the alignment problem, we formulate protein threading as a linear problem with a set of constraints that can be solved using a technique called Integer Programming. In this method the linear equation is constructed with two sets of variables, the x set, and the y set. The optimal set of variables to minimize the objective function created by these variables, their coefficients, and the constraints involved is solved by linear programming. A majority of the time linear programming provides an integer based solution for the optimal value. In the other cases, a standard technique called branch and cut can be used to find an integer solution. First we introduce a few variables. $X_{i,l}$ represents the single residue energy function of core i at position l. $Y_{i,l,j,m}$ represents the residue pair energy function for core i and j at positions l and m, respectively. Each of these energy function values has an associated binary value, one if the energy element is part of the alignment, and zero otherwise. This is done by constraining the coefficient value between zero and one and forcing the solution to be an integer. Each energy function coefficient is associated with a specific variable that governs if it is or is not in the final solution. X is associated with x, and Y is associated with y. Using the x and y variables,

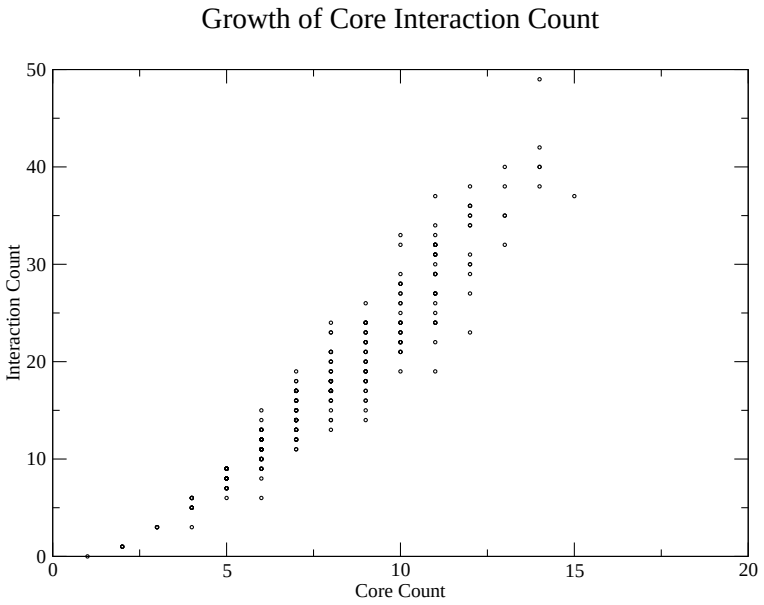

Figure 5.4: The growth of the number of interactions as the number of cores increases.

the relationship between the X and Y values can be mapped. Given these variables and values, the objective function to optimize is given as

$$\min_{x,y} \sum_{i \in M} \left(\sum_{l \in D[i]} x_{i,l} X_{i,l} \right) + \sum_{(i,j) \in I} \left(\sum_{(l,m) \in (i,j)} y_{i,l,j,m} Y_{i,l,j,m} \right) \tag{5.1}$$

This objective function minimization is then constrained by 3 sets of equations.

$$\sum_{i \in D[a]} x_{a,i} = 1, a = 1, 2, ...M \tag{5.2}$$

$$\sum_{j \in R[a,b,i]} y_{a,i,b,j} = x_{a,i} \text{where } (a,b) \in C \tag{5.3}$$

$$\sum_{i \in R[b,a,j]} y_{a,i,b,j} = x_{b,j} \text{where } (a,b) \in C \tag{5.4}$$

where $D[i]$ is the set of position for core i, $R[j|i,l]$ is the set of positions for core j given that core i is at position l, and I is the set of core pairs that are connected, and N is the number of cores. Once the coefficients of the objective function have been calculated and the matrix of constraints equations has been formulated, the problem can be passed to a integer programming solving package like COIN [Lougee-Heimer, 2003].

5.2.2 Energy Functions

The energy functions used to create the objective function co-efficients can be found in section 2.1.1

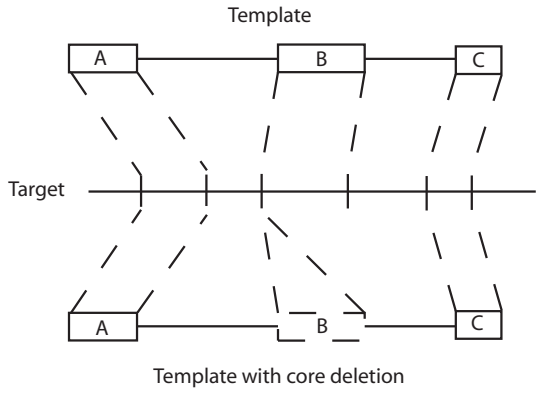

Figure 5.5: Core B, aligned to the same position: in the regular state and in the deleted state.

5.2.3 The Deletion State

We expand the above Integer programming model by adding additional positions to the $D[i]$ that represent aligning core i to the target, but deleting it. When a core is deleted, it takes up no space on the target sequence. This is illustrated in Figure 5.5.

This means that there are twice as many target residues that the cores can be mapped to. Deleted states can be denoted with d.

$$D[i] = \{0, 1, 2, 3, ..., L-1\} \qquad \text{Original Model} \qquad (5.5)$$

$$D[i] = \{0, 0d, 1, 1d, 2, 2d, 3, 3d, ..., L-1, (L-1)d\} \quad \text{Core Deletion Model} \quad (5.6)$$

When a core is deleted, the associated pair interaction energies are set to zero, save the energies connecting the cores that occur sequential before and after the

deleted core. The pair energy with the core immediately preceding the current core is the result of the loop region dynamic programming alignment as calculated before, as illustrated in figure 5.6. This figure represents the typical alignment graph, a move down is a template deletion, while a move to the left is a target deletion, a diagonal down and left is a match. The core placement determines the alignment patterns in certain portions of the alignment chart, the rest, between the blackened cells, can be determined by regular dynamic programming. The pair energy for the core after the deleted core is calculated by dynamic programming beginning with the deleted core. We also add a static penalty for each core deletion that is stored in the X coefficient. This is to reinforce the idea that secondary structure element deletion has more effect on the protein structure and must be avoided unless necessary.

5.2.4 Terminal Deletions

When threading a target against a template that is much larger than it, it is possible to run into a scenario where the total space taken up by the cores is larger than the sequence of the target. In these cases, unless allowances are made, there may be no feasible solution for core placement. There are also many cases where a core at the very beginning or the very end of the protein template does not properly align to the target. These elements may be part of the 'fringe' structure, and not core structure of the protein. In these cases, it is necessary to model N-terminal and C-terminal deletions. In previous versions of the model, this was handled by inserting blank 'dummy' residues on each side of the protein. These residues have energy

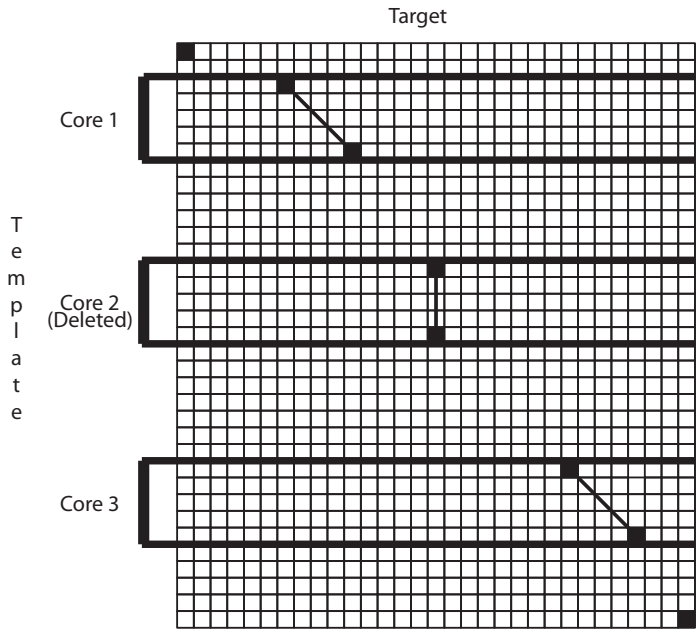

Figure 5.6: Core deletion, as seen in an alignment matrix

functions that only produce zeros. And once the alignment is calculated they are removed, the aligned region is treated like a deletion. It is important to realize that because all of these dummy residues have no energy there is no way to distinguish between them. And because this region will be deleted, it doesn't matter if the cores overlap. This means that all of the dummy residues at a given end of the protein are redundant and can be consolidated down to a single state. This creates two deletion states, the N-terminal deletion state and the C-terminal deletion state. Thus, the size of the target sequence included the dummy residues that surround it, can be greatly decreased. When describing this in terms of the objective function, all the terminal states that would have been described as $X_{1,-50}$, $X_{1,-49}$, $X_{1,-48}$, ..., can be described with the single term $X_{1,Nterm}$. The same applies for the residues on the C-terminal side of the protein. This simple adjustment to the model makes it easier to model terminal deletions of the template while using a fraction of the memory previously needed.

5.3 Results

To see if our core deletion model can improve the sequence-structure alignment, we tested our core deletion model on a few protein pairs that have different core numbers. We have found that the alignments improved with the addition of the core deletion model. In these cases, the core deletion correctly identified secondary structure elements that did not exist in the target protein structure.

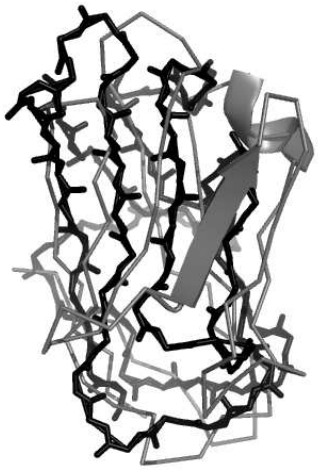

Figure 5.7: An example protein structure alignment of d1cwva4 and d1cdy_1.

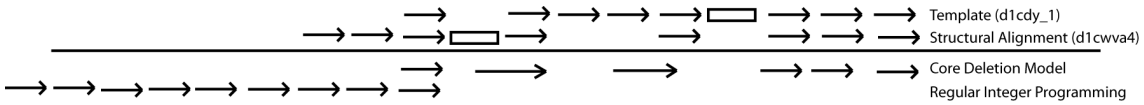

Figure 5.8: An example of an alignment created by the old method, and one created by the core deletion model.

In the example of the d1cwva4 to d1cdy_1 alignment, shown in Figures 5.7 and 5.8, two β-strands and a α-helix had to be removed in order to find the correct alignment. In this case, PsiPred did not correctly predict the α-helix in the target secondary structure sequence. The original integer programming had no way to deal with internal deletions, and thus found it to be more energetically favorable to slide all the cores to the N terminal region. As illustrated in Figure 5.8. Using an structural alignment generated by FAST [Zhu and Weng, 2005], we score the alignment generated by the various threading methods. The residue score is (0.5)k where k is the distance from where the residue was aligned and where is should have been. It is 0 if $k > 3$. The points are reported as a percentage out of the total possible. In this sample, the Dynamic Programming score is 11%, but the Integer Programming model is 0%. This is because the only way that model could deal with the cores that needed to be deleted was to delete them off the N terminal edge. But when we apply the core deletion model, the accuracy goes to 30%.

We have also tested this method against a protein pair previously studied by Kolodny et al [Kolodny et al., 2006], shown in Figure 5.9. This alignment is between a DNA mismatch repair protein PMS2, and a protein with unknown function, TM1457. In this test the identified protein, PMS2, acts as the template structure. It has 5 secondary structure elements that are not part of the TM1457 protein, as illustrated in Figure 5.9. This is a perfect example to test the utility of the core deletion model. First, it is a demonstration of the internal secondary structure deletion that we are trying to model. Next, the data about the target sequence, TM1457, is

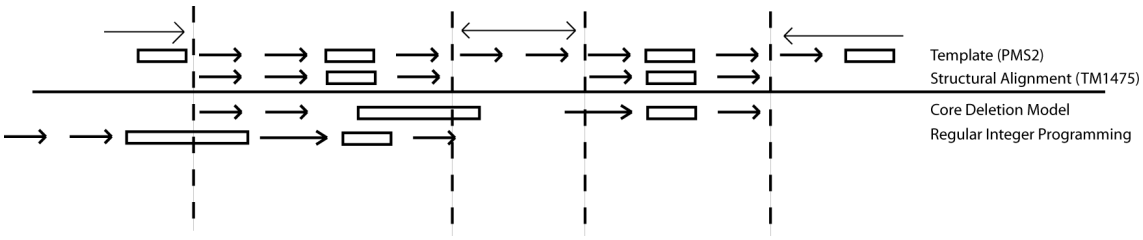

Figure 5.9: The alignment between TM1457 and PMS2 [Fischer et al., 2003].

very limited. The Psi-Blast frequency matrix held no data, as there where no hits in the NR database used to create the mutation matrix. Also, PsiPred, mistakenly predicted the secondary structure of the 4th and 5th secondary structure elements, a α-helix and β-strand, as a single α-helix. That means that a good alignment would have to come by analyzing structural information, such as the singleton energy and the Dfire residue pair energy. Using the previously described alignment scoring system, dynamic programming gets 32%. Because of the limitations of the Integer Programming model finds no correct alignments at all. This is caused because the alignment must include the 6th and 7th cores, which should be deleted. However, when we use integer programming with the extended core deletion model the alignment accuracy jumps to 54%.

The original form of integer programming fails, because it cannot delete the cores in the middle of the sequence. Dynamic programming is able to model the core deletions, but because it cannot optimize the alignment using the Dfire energy it has less information to create the correct alignment. Integer programming, with

the core deletion model, is able to model this problem more realistically, and utilizes all structural information in order to properly model this distant homologue.

5.4 Discussions

With this work, we have demonstrated a tractable model for aligning protein sequences to a template using residue pair energy functions and allowing for core deletions without compromising any of the features in the original integer threading model. And while this technique increases the total amount of memory needed, we have also proposed the idea of terminal deletion states that remove redundant blank residues. This helps compensate for the increase in memory by reducing the memory requirements for other parts of the problem. Because the original Integer Programming model is a subset of this new formulation, this technique should be able to solve all of the alignments original model was able to. Besides memory constraints, the only thing that would separate these two models would be the possibility of 'false-positive' deletions. With improvements to the energy functions, that problem should be minimized. Now that there is a working model, we can begin to formulate more complex energy functions to better represent the likelihood of core deletions. In this work, we utilized a simple static penalty for core deletion. It would be reasonable to argue that secondary structure elements on the surface, or β-strands on the edge of a β-sheet, are more likely to be deleted than the ones in internal regions. A separate deletion penalty could be applied to each

of the cores, giving a higher penalty for deletion of internal cores. There could also be mutually dependent deletion energies, in which pairs of elements that are more likely to be deleted together than individually. These ideas can easily be added to the model without increasing the algorithmic complexity or the amount of needed memory. In order to implement these more complex energy functions, all that would have to be changed would be the coefficients for the variables solved by the integer programming. There would be no increase in algorithmic complexity or memory requirements beyond those outlined in this paper.

5.5 Conclusion

We believe that this new core deletion model will allow us to better map the conserved sub structures of proteins, removing secondary structure elements as necessary. We believe that the addition of the ability will greatly improve the range of applications for the threading technique.

Chapter 6

A Genome Scale Protein Structure Prediction Pipeline Using Automatic Parallelization

Some of the results below have been previously published as:

Guo, J-T. Ellrott, K. Chung, W.J. Xu, D. Passovets, S. Xu, Y. "PROSPECT-PSPP: an automatic computational pipeline for protein structure prediction", Nucleic Acids Research 32: 522-525 (2004)

Kim, D. Xu, D. Guo, J-T. Ellrott, K. Xu, Y. "PROSPECT II: Protein Structure Prediction Program for the Genome-scale Application", Protein Eng. 16(9), 641-650, (2003)

The previous chapters of this dissertation have concentrated on improving different aspects of the threading paradigm in order to improve the ability to identify distantly related proteins that are in the same protein fold family. In this chapter we concentrate on connecting threading to a larger framework of tools for protein structure prediction. These tools provide the information and guidance needed to run threading in an appropriate manner. Also, we concentrate on the automation and parallelization of the prediction process. These improvements seek to create a system that is able to handle protein structure prediction at a genomic level.

6.1 Introduction

The rate of DNA sequencing has increased dramatically in the past decade, and continues to grow at a phenomenal rate. With various projects such as GenomesToLife and CAMERA (Cyberinfrastructure for Advanced Marine Microbial Ecology Research and Analysis), the number of sequenced genomes or gene collections have grown dramatically. As of autumn of 2007, there were over 600 prokaryotic genomes available for public download at the National Center for Biotechnology Information (NCBI) website. This is due to the fact that the technology for determining genomic sequences is becoming cheaper and more efficient. Structure determination techniques, on the other hand, have not seen the same strides in performance improvement. This has created the need for computation prediction of structures and for systems that can handle protein structure prediction at the genome level.

The mantra of proteomics is that sequence determines structure which determines function. Sequencing technology has given us the ability to extract a whole genome sequence. Genomic sequence analysis can help to determine gene function by looking for conserved sequences. However, proteins do not need to share much sequence similarity in order to have the same structure. In these cases, sequence homology can not be used in order to determine protein function. By design, threading is able to find proteins with similar structures but very little sequence similarity. Because of this, threading can be a valuable source of information for genome annotation. However structure prediction programs are not typically designed to work efficiently at a genome scale.

There are a multitude of aspects and tools related to protein structure prediction. These tools include Psi-Blast, secondary structure prediction programs, membrane protein prediction utilities, and signal peptide identification methods. There are dozens of tools that have been used to analyze protein sequences to reveal more about their structure. Because most of these tools are independently developed, there is frequently a lack of direct compatibility in usage and in file formats. We have proposed a Protein Structure prediction pipeline for the purpose of tying together these tools in an automated fashion. These tools are controlled by a set of logical rules in order to make sure that the correct tools are run for the correct situations.

A protein structure prediction pipeline is designed to start with the amino acid sequence. While there is some configuration that can be done in the way of parameters for various tools, ideally it should take no more then entering the sequence

and pressing 'Go'. With this as a designed goal, applying the pipeline to an entire genome becomes possible. The only required input is the amino acid sequences for each of the predicted genes.

The concept of an organized workflow is a developing idea in the field of computer science. A workflow is a deterministic set of connected operations that occur on data that produce output or results based on that data. These workflows are defined by a set of operators to perform the work, the organization that defines the movement of data from one operation to the next and the algorithms that schedule these operations. A workflow can be modeled as a graph where the operations are represented by nodes and the flow of data between them is represented by the edges. Many fields, such as bioinformatics, utilize many separate and independent tools in order to analyzes information, perform calculations, and model data. A workflow ties these separate tools together into a well defined set of steps that must occur for every new piece of input data. This is particularly useful in large scale problems such as genome research where the sheer volume of data would overwhelm a human operator.

For our research, we define a pipeline as a subset of the workflow problem. The primary feature of a pipeline is that it can be represented by an acyclic directed graph. In this way, a piece of data starts at the beginning, flows through a series of steps, and eventually reaches the end, much like water flowing through a pipe. A node operation cannot occur until all of its inputs have been filled. We also define that the balance of data between input and output sets must be one to one. This

means that one set of input values produce one set of output values. Given these constraints, data moves through the graph in a predictable fashion. This also means that the one to one relationship that defines the individual nodes is inherited by the larger graph.

While the pipeline is acyclic, that does not mean that it is a straight line. There can be many parallel paths that data must flow along simultaneously before re-merging at the end. It is important to identify these parallel operations, as these parts of the operation graph can be dealt with simultaneously. Beyond operational parallelism, there is the notion of data parallelism. As mentioned, a pipeline is acyclic, but this does not mean the concept of a loop is forgotten. Frequently in programming, a loop is written so that the same operation can be performed on each member of an array, one after the other. These types of loops could be said to be performing vector operations. Because each of these operations are independent, the order in which they occur is not important. If these points can be identified, then the data can be dispersed and operated on in parallel.

We have created an operational environment that automatically parallelizes a user defined pipeline in a distributed memory system. Given the structural graph that describes the flow of work along the pipeline, and the ability to serialize data, parallelization becomes transparent to the user. Our pipeline model works in a operation and data parallel fashion.

We have utilized particular features of the programming environments serialize data as it passes between operation nodes. This enables the parallel machine to exist

across different memory addresses, such as a cluster environment or even machines spread across the internet. This also means that the state of a pipeline can be captured and restarted. All of this functionality is transparent to the user. The user does not have to program to fit any model of parallelism, rather they merely have to fit the graph model of a pipeline. We believe that this model of programming will be more accessible for novice programmers unfamiliar with the ideas of parallel processing.

This new tool, called Parallel Integrated Python Pipeline EnviRonment (PIPPER), has enabled us to quickly implement a protein structure prediction pipeline, similar in function to PROSPECT-PSPP [Y. Guo, 2004]. The primary differences between the two is that the PIPPER script of the pipeline was significantly easier to implement and debug. At the same time the end product was faster and more able to take advantage of the full computational power and parallel nature of a distributed memory computer cluster.

6.2 Materials and Methods

6.2.1 Pipeline Architecture

Each of the amino acid sequences must undergo that same process of analysis. The pipeline consists of the following stages; Pre-processing, Collection, Triage, Fold Recognition, and Structure Prediction.

The Pipeline Manager

1. Pre-processing: Determine basic information about protein sequence, such as locations of trans-membrane regions or signal peptides that would be cleaved.

2. Collection of functional/structural information: Use prodom blast to determine likely domains.

3. Protein triage: Determine if the protein has transmembrane regions or domains that are close to existing determined proteins.

4. Protein fold recognition: Run protein threading program in order to find distant homologues.

5. Protein Structure Prediction: Use protein modeling program to determine atomic coordinates.

Pipeline Tools

Individual Tools

1. SignalP

   SignalP [Bendtsen et al., 2004] is a program to identify Signal Peptides. Signal peptide are usually cleaved off and are not part of the end protein structure. Thus a signal peptides will be trimmed off of the target sequence before the bulk of the structure prediction process begins.

2. Prodom

Prodom [Corpet et al., 2000] is a comprehensive database of protein domains. Domains can be identified by a sequence search program, such as BLAST [Altschul et al., 1997] by scanning against the database of known domains. Domains are mostly independent in their folding processes and can be predicted independently. From a computational perspective smaller sequences are less of a challenge and each domain can be analyzed with in parallel.

3. TMap

   TMap is a program designed to identify membrane proteins [Persson and Argos, 1994]. Since membrane proteins can not be predicted using threading techniques they will be filtered out before prediction. There are currently new methods being develop to deal with membrane protein structure. When these techniques are ready, they can be added into the pipeline.

4. PSI-Blast

   PSI-Blast [Altschul et al., 1997] is a well referenced program for searching databases for similar sequences. PSI-Blast can be used initially to scan the Protein DataBank in order to search for close homologues that have already been characterized. If such a homologue is found, the threading portion of the pipeline can be skipped, and instead protein homology modeling can be done using the existing model. PSI-Blast is also used to create a Point Specific Score Matrix (PSSM) which is the basis for the mutation energy function utilized during threading. This PSSM is also the source of information for PSIPred.

5. PSIPred

   PSIPred [Jones, 1999] is a Secondary Structure prediction tool. It is based on neural network recognition system that classifies windows of amino acids into three categories, Coil, α-helix or β-sheet. It is able to correctly classify a residue to one of the three groups about 85% of the time.

6. Prospect

   Prospect [Kim et al., 2003, Y. Guo, 2004, Xu and Xu, 2000] is a protein threading tool. My improvements to its methods and energy function have been outlined in the previous chapters.

7. Modeller

   Modeller [Sali and Blundell, 1993] is a homology modeling program. It takes a sequence/structure alignment and produces a set of atomic coordinates.

The logic of the pipeline starts with pre-processing. This first occurs with signal peptide processing. Because signal peptides are cleaved, they should not be included in the sequence being submitted for structure prediction. This can be handled by the tool SignalP.

Homology modeling and threading typically only work for globular proteins. Membrane proteins lack a sufficient library of known structures to enable wide representation in a threading library. Also, the threading energies have been tuned to deal with globular proteins, and would have to be redesigned for a more accurate representation of energies in membrane proteins. While there is ongoing research

for membrane protein structure prediction, and the time of the pipeline's design the technology was not yet practical. For this reason, if a protein identified by as a membrane protein, the pipeline stops. After the elimination of the membrane proteins, domain prediction is the next step.

6.2.2 Pipeline Description

In order to enable this environment, a user is required to describe the structure of the workflow using a XML formatted description and provide a script of defined user code that provides the functions that will be used at each operational node. The most basic element of the pipeline description is the concept of a node. A node is an operational point in the pipeline graph and it is associated with a single function call. The description maps the inputs to the outputs. Because the pipeline is a directed graph, all edges in the graph are described from the node where they lead to. Thus, the inputs of a node are mapped to the outputs of the nodes that provide them with data. Eventually, if we trace these directed edges back they would lead to inputs with sources that lead outside of the graph. These are the inputs that allow for user input data.

Nodes must have balanced input and output. For every element set that flows in, the exact same number of element sets must flow out. Array dispersal and reassembly occur completely inside a node. This characteristic is enforced to make sure that parallel pathways do not produce different numbers of outputs. Imbalanced

output counts would lead to non-deterministic consequences during data reassembly or input collection.

An input node can define that that incoming data must be an array of independent elements. With appropriate annotation, the pipeline management system will automatically disassemble and disseminate the array. This allows for the data parallel operation of the pipeline. While the input/output relationships must be 1:1, that does not mean that a single input value can't produce a vector of output values. If a given function takes an integer as an input and returns an array of integer values as a result, it still fulfills the 1:1 requirement but only if the entire array is treated as a single element when connecting to the next stage of the pipeline.

A nodeset is an expansion of the original concept of a node. A node has inputs, outputs and an associated function call. A nodeset has the same, except the associated function call is actually another graph of operation nodes that is completely contained within that node. In essence the nodeset is an isolated subset of nodes in the graph that have a single input set and output set. This subset of the graph can be set to operate on each of the independent elements of an input array. Thus an entire section of the pipeline can be set to be data parallel. This allows for a number of different operation nodes to be visited by data before the array recombination occurs.

6.2.3 Algorithms and Data Structures

The pipeline algorithm operates in both a data and operational parallel fashion. Operational parallelism occurs when there are multiple paths in the graph from one node to another. The data parallelism is not as easy to see. Data parallelism occurs inside a defined nodeset, where the input is an array of independent elements. The necessary one-to-one relationship in the number of inputs and outputs in a nodeset keeps the data dispersal and recombination even. A recombined array will contain the same number of elements as the original array before it was split.

Each node in the graph is made of three distinct parts, the input set, the function node, and the output set (See Figure 6.1). The input set scans the output nodes to which each of the input ports is attached. The input set is also responsible for analyzing incoming arrays and splitting them into sets of independent elements. The function node is responsible for checking the input set to make sure a joint set of data spreads across all of the input nodes. If the data in port A is from calculation 1 and the data in port B is from calculation 2, no operation can be performed until the data from one of the calculation sets spans all of the ports. The output set is responsible for collecting data until all required elements are available to be combined. Usually an output can pass data along immediately and the combination is only necessary if the output is for a nodeset that operated on a split array. If an array combination is needed then data is queued until all separate calculations for the array have been done. Once all of the data needed for the total array has

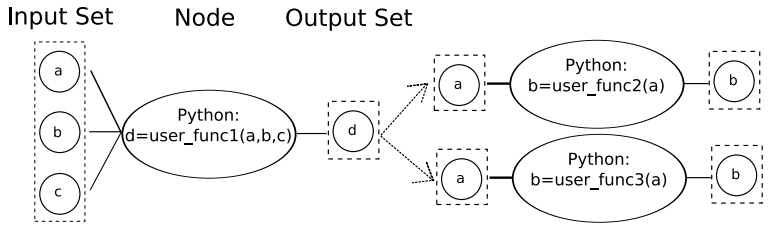

Figure 6.1: An Example of Operational Parallelism

been collected the queued data is recombined into an array in the same order as the input data and then passed along.

The scheduling algorithm is a depth-first-once-visit search of the tree pipeline from each of the output ports. Along the search, the work that needs to be done is determined for each of the visited nodes. The algorithm catalogues a list of operations that are ready to occur at the time of the tree scan. These operations include: moving data from output ports to input ports, splitting arrays that have arrived at input nodes, calling functions at nodes where all of the dependent data has arrived, and recombining arrays that have collected in output nodes.

During initial setup of the pipeline the tree is scanned with a depth first, once visit, scan that increments a reference count at every I/O node that is scanned (before the 'already visited' feature is checked). This is a quick way to map data dependencies and allows for a simple garbage collecting scheme. A port will copy a piece of data for every incoming edge. If a port has N incoming edges, after the data has been fetched N it will deallocate the original copy of the data. Thus duplication and deletion of data on the interior of the graph is taken care of.

6.2.4 Programming Environment

The selection of language for this problem is very important because of the necessity of serialization. Serialization is the ability to take complex and structured data from the internal memory of a piece of software and turn it into a sequential series of bytes. For this reason the scripting language for our pipeline implementation is Python.

Python allows for a variety of data types to be automatically serialized. The basic data types include the 'None' value, Boolean values, integers, long integers, floating point numbers, complex numbers, normal and Unicode strings. More complex values include tuples, lists, sets, and dictionaries (the python equivalent of a hash table), so long as these complex types are comprise of serializable types. It will also allow the serialization of object data, again so long as the class is comprised of serializable data. Python is able to handle language abstraction, such as objects that wrap links to compiled C/C++ code. These objects are not serializable, as the pointers and binary data will lack proper annotation for serialization. This serialization is handled by an included module called 'Pickle'. This module takes a user data structure, decompose it into a text description that can be stored on disk or sent across the network. This module then recomposes the text data back into a functional object. It is this ability to serialize data that enables Python scripts to be seamlessly dispersed into a cluster of machines and easily pass information between them.

The management level of the program is written in a C/C++ environment. The Python development kit allows of the embedding for the Python interpreter inside of a C/C++ program. It is inside the C/C++ program that the user pipeline description is analyzed and the data the flow across the pipeline is managed. Essentially, the manager calls the appropriate user written Python code and then inspects the interpreter's memory environment to catalogue and access user generated data. The manager's actions are defined by the pipeline structure description provided by the user.

The parallel features of the program are implemented in MPI. Each of the separate python interpreters runs in a different memory space. This means that all data passing between functions must occur through the ports defined in the pipeline structure description. This also means that it is not necessary for functions to be reentrant.

The structure description of the pipeline is written in XML in order to maintain portability and ease integration with other tools. A 'node' entry represents a single python function call, while a 'nodeset' defines a set of linked nodes and nodesets. The tree like structure of XML is used to define the recursive nature of nodesets. The data parallelism is designed to be multidimensional.

6.2.5 Parallel Aspects of Data Transfer

Parallel processing has been implemented with a distributed memory environment in mind. A set of M processes starts up on a group of connected CPUs and each

loads the pipeline description. A single node is designated the master while the other $M - 1$ nodes are considered to be workers. Only the master node is aware of the full state of the graph at any one time. However, the master node does not hold the actual data in memory, rather it holds data stubs that point to the processes that hold the data. The master's job is to scan the current state of the pipeline tree and coordinate data transfer between the worker processes and instruct them which operation nodes need to be called and when to collect them.

The hardest part of scheduling is the problem of call blocking during the python functions. As currently implemented, a worker node calls the user python functions in a synchronous fashion. This means that when the user code is active, the other data held by that process is inaccessible. A user function cannot be called until all necessary data is held by a single process. The manager coordinates the data processing by determining the process that holds the largest percentage of data of a given operation and if the data not held by that process is held by processes that are not currently blocked. If all of the data is available, the master instructs the other workers to send data to the selected worker. Once all necessary data for a particular operation is held by a single process, the master issues a command to perform the operation to that worker.

The data parallelism nature of the pipeline creates a need to maintain a namespace for variables outside of the user's original naming scheme. This naming scheme is tree like in nature. Every array dispersal takes a leaf in the tree as an input and

populates it with a set of children nodes. Each child node represents a single element of data in the dispersed array. At the point of the array recombination, the set of the children being recombined are watched until all of the operations have been completed. Once all of the completed data leaves have completed, the data is recombined into a single array and the parent becomes a leaf again.

The naming system used to keep track of the data parallelism is referred to as the 'split space' naming. In order to maintain the ease of communication across the network, the split space name is kept in a simple string format. Eqn 6.1 demonstrates an example of a split space naming.

$$0; 4/8; 15/16; 23/42 \tag{6.1}$$

In this case, the data parallelism has reached a three dimensional split. The root input is the $0th$ element. After that, each of the splits is denoted by a fraction listing, separated by semicolons, with the element number in the numerator and the total number of elements in the denominator. The first data split produced $8th$ element, of which the example element is the $4th$ element. From there, that element of data was split into 16 elements, of which the $15th$ element was again split into 42 elements. This also means that before the data element '$0; 4/8; 15/16$' can be recombined, 42 elements must collected.

6.2.6 Scheduling Parallel Tasks

In a majority of parallelization problems, there is what could be termed, an expansion and contraction of resource needs. Certain points of calculations are easier parallelizable, and thus can be run on several computers simultaneously. These periods would be the expansion stages. In other cases, calculations are best suited for a single machine. These would be be the contraction stages. When scheduling parallel tasks, a naive solution may cause the causes the expansion contraction patterns illustrated in Figure 6.2. Because resources are assigned in static blocks, this expansion and contraction can causes under utilized systems. The size of resource allocation determines how wide the expansion of parallelism can be. But in times that the system contracting there will be a large number of resources that sit unused.

The pipeline topology allows for a method to track work that needs to be done. At any moment, the pipeline manager should be able to determine a piece of work that needs to be done for the resources that are available. Ideally this will allow a system of parallelism seen in Figure 6.3, where expansion and contractions are spread out to form a constant total usage over time.

To avoid 'explosive' expansion during calculations, a two-strategy approach has been employed. First, only a small set of calculations are considered 'active' at one time. The set size is usually $10N$, where N is the number of worker processes. This subset of active jobs are tracked and prepared for calculations, and it is from this set that actual running calculations are started. A job must be prepared before it

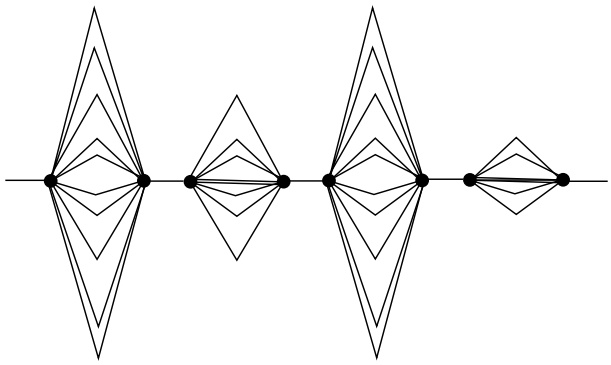

Figure 6.2: The expansion and contraction of parallel tasks

can run. Preparation involves moving all needed input data to the same worker process. Tracking and migrating these active tasks can become memory, CPU and network intensive. This is the reason to keep the active set a limited size. Work requests that do not become part of the active set will simply be regenerated upon the next pipeline tree examination. The second strategy used to spread calculations is a scoring system that prioritizes certain requested to be added to the active set. This is done by creating a hash table of data element parent split strings (Eqn 6.1) that counts the number of occurrence seen in the array collection sections of the pipeline. Work requests that have the same parent split as data that is waiting to be merged are prioritized. Thus array splits that are half done are prioritized over ones that are just starting out. In addition, work nodes are sorted by depth to the end points of the pipeline. In this way, work that closer to the end of the pipeline is prioritized over work that is closer to the beginning. This creates a system where one array split is calculated and merged before the next split calculation starts.

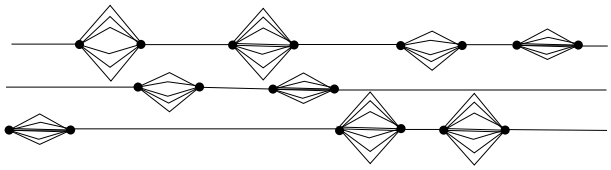

Figure 6.3: The expansion and contraction of parallel tasks spread across time

6.2.7 Example Usage of Pipeline Programming

The Mandelbrot Fractal

The installation and setup of the protein structure prediction pipeline require a 3GB installation of multiple databases and separately licensed programs. In order to provide a simple and easy to install example of the automated parallelization techniques we have included an example implementation of a program to generate the mandelbrot fractal. This is a classical fractal description that maps a set of points in a complex plain by counting the number of iterations in which the complex quadratic polynomial $x^2 + c$ remains bounded. The calculation of each of the pixels is independent, thus the problem has the possibility of 2-dimensional parallelism. However, in the case presented, only 1-dimensional parallelism was employed. The code for this example is listed in Supplemental Material. In this implementation, the input to the pipeline is the desired resolution of the pixel calculations. Parallelism is done in a one dimensional manner. A call to the function 'x_range' produces an array with the x-coordinates for each of the rows to be calculated. This array is then broken up into independent elements, and the pixel calculations are done row

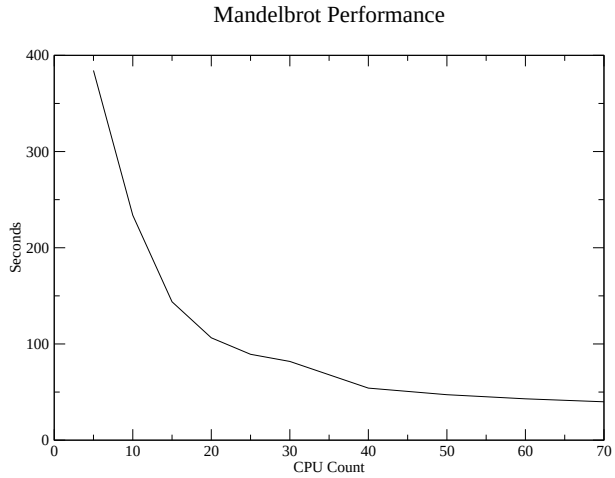

Figure 6.4: Computational Performance of Mandelbrot calculation vs. Cluster Size

by row. Once all of the rows are done, the arrays are combined into a new output array and passed to a printing function.

Performance testing was done on a 1000x1000 grid of points, with each pixel calculation allowed a maximum of 2000 iteration before exiting. The testing was done on a cluster of 3GHz Xeon processors connected by Gigabit Ethernet. The results of this test can be seen in Figure 6.4.

Protein Structure Prediction

The protein pipeline starts first at the genome level. Each protein amino acid sequence is a separate and independent structure problem. This is the first level of data parallization. For each protein sequences, the first step is to partition the protein into domains. Each of those domains represents a separate data parallel portion of the pipeline. Each of the domains can then be scanned against the existing PDB

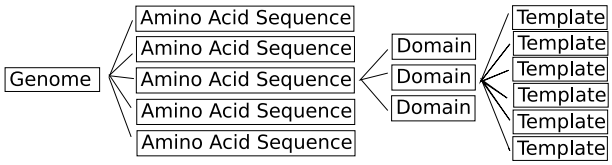

Figure 6.5: The data parallel nature of the protein structure prediction problem

database for easy to find hits. Domains that are predicted to be primarily composed of transmembrane domains are stopped after identification, as a majority of structure techniques are geared toward finding soluble proteins. Each candidate protein domain sequence is then profiled against the Non-Redundant protein database provided by NCBI. Secondary structure prediction is done by PsiPred [Jones, 1999], and Protein Threading is done by Prospect [Ellrott et al., 2006, Kim et al., 2003, Xu and Xu, 2000, Xu et al., 2001b]. Because protein threading is a database search, it is also parallizable. Finally, once protein folds have been selected, Modeller [Sali and Blundell, 1993] can be run to determine atomic coordinates.

The data parallel nature of the protein structure prediction problem is illustrated in Figure 6.5, and visualized in the graph model in Figure 6.6. The typical prokaryotic genome encodes on average about 1,500 to 5,000 protein sequences. A protein can have up to three or four domains and the template library is composed of a representative set of about 4,500 proteins. As a rough estimate that is between 3,000 and 15,000 separate domains that can be calculated independently.

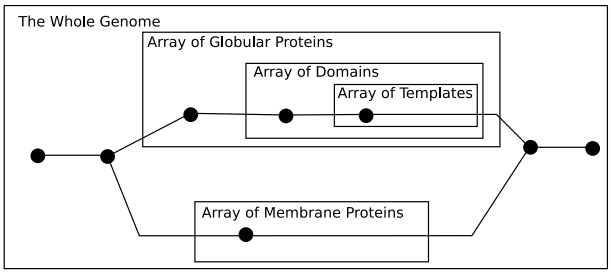

Figure 6.6: The data parallel portions of Protein Structure prediction, as viewed on the pipeline workflow graph

6.3 Results

6.3.1 Genome Applications

30-40% of genes in a newly sequenced genome do not have any functional assignments using sequences based approaches like BLAST [Altschul et al., 1997]. The use of protein structure information to improve alignments of distantly related proteins, i.e. protein threading, can help to broaden the ability to predict the function of newly sequenced proteins.

For our tests, the protein sequences were split into domains and each of the domains was checked by the trans-membrane identification programs. Domains predicted to be globular were then scanned against the existing amino acid sequences in the Protein Data Bank. Protein sequences that did not have PDB homologues were threaded against a representative set of protein structures. Their protein sequence/structure alignments were then assessed using a gradient boosting based method [Jiao et al., 2006] referred to as AA_{pred}. Using a testing set of 724 proteins,

Table 6.1: AA_{pred} tested Performance

AA_{pred} cutoff	Sensitivity	Reliability
20	17%	25%
30	5%	60%
40	3%	90%

which fold families identified by SCOP, we have sampled the prediction sensitivity/reliability curve, shown in Table 6.1, for this scoring scheme. In this table the sensitivity is the percentage of fold level, and greater, pairs that were successfully detected out of all the defined pairs. The reliability is the percentage of selected set that we correctly identified. The set of $AA_{pred} > 40$ is a set of proteins which have extreme confidence in prediction but were not detected using homology modeling. The results of our scans can be seen in Table 6.2.

6.4 Discussion

The underlying bottleneck in parallel implementation of the current version of this pipeline management software is the blocking nature Python calls. This can be addressed by making the Python operation asynchronous by placing it in a separate thread to do the calculation. The other thread can then continue to monitor for data requests and maintain communications while the user command executes. This is not trivial however, as serialization is handled by the Python interpreter, which is already engaged in processing the user code. We need to either verify that the

Table 6.2: Protein Structure Prediction Pipeline Results

	Haemophilus influenzae	*Synechococcus*	*Pyrococcus abyssi*
Proteins	1657	2519	1896
Globular Domains	2677	3530	2542
Trans-Membrane Domains	840	1101	981
PDB Homologues	1349	1028	914
Prospect Scans	1372	2558	1660
$AA_{pred} > 20$	212	587	329
$AA_{pred} > 30$	72	227	114
$AA_{pred} > 40$	25	97	37

Python interpreter code is reentrant, or pre-cache a serialized copy of all the variables held by that process before executing the user python code is called. The current implementation only serializes data when network transfer is necessary. Pre-caching could lead to memory requirements doubling, and larger latency for user function calls.

Given the framework that has been developed, there are many advancements that could be made to this technology. These advancements have not yet been implemented, but there is no technological reasons to prohibit their development. The most notable of these features is the possibility of 'checkpointing' work. Because the intermediate data between function calls can be serialized on demand and it's current state is well defined by its position in the pipeline graph, it would be very easy to build a system that captures the state of the graph at any one moment and stores it to disk. In the world of high performance computing, there is a distinct need for data checkpointing. The more computers involved in a calculation, the more

likely that one of them will crash during the run. If there is a crash, it is desirable to restart the computation from a checkpoint rather then completely restarting the job.

Another possibility for advancement is the management of a widely distributed computation. Currently PIPPER has been designed to work in an MPI environment with a shared file system, but a web service orientated method could also be possible. It is entirely possible to communicate instructions and code with a remote service system like SOAP. This could be used to help coordinate a massively distributed workflow, similar to projects like Folding @ Home [Shirts and Pande, 2000].

6.5 Conclusion

From the user's perspective, aside from the benefits of automated parallism, one of the most exciting features of PIPPER is the removal of code customization typically needed. There is little to no 'product lock-in' demanded by PIPPER. PIPPER requires the user write functions that can be mapped together using the described pipeline graph topology, and then describe this topology in XML. But once that work is done, those functions can easily be reused in other systems that completely excludes PIPPER.

We believe this pipeline technology is an exciting step in distributed processing. It enables users who have experience with programming scripts to quickly and easily

setup parallel programming systems with little to no knowledge about the principles of distributed computing and parallel processing.

There are many different bioinformatics and computational biology tools that are being scaled up from working at the level of a handful of genes to working on entire genomes. Single genomes are just the beginning of data explosion. As metagenomics becomes more popular, the need to easily parallelize work loads will only become more acute. We have shown that this technology has been quickly and easily applied to the problem of genome level protein structure prediction. The framework quickly adapts to take advantage of the operational and data parallel natures of the tools that need to be run for data analysis.

Chapter 7

Conclusions and Perspectives

The work in this dissertation has helped to advance several aspects of protein threading. Starting with energy function, the idea of the variable-deletion penalty is based on the idea that applying protein family deletion information to a protein sequence/structure alignment could improve alignment accuracy and fold recognition. This was shown to be very successful in improving both alignment accuracy and fold recognition. The work with structural clustering based on threading results found that available information could be applied to alignment refinement. I have taken the existing method for Linear Integer Programming based protein threading and shown that by adding the simple concept of an 'imaginary residue position', certain limitations of the algorithm were easily removed. And finally I showed that by taking several existing tools and integrating them together into a workflow one could more successfully utilize protein threading. We saw that by analyzing the graph of data dependencies in the workflow we were able to automatically disperse

data and schedule work in order to maximize effective resource usage and minimize runtime. This automatic parallelization enables protein threading to be carried out at the genome level.

Personally, some of the most exciting technologies developed for this dissertation are the methods for automatic pipeline parallelization. These techniques have the greatest possibility for re-application to other fields. At its core, the technology is not specific to the protein folding problem. Techniques to make parallel programming for distributed memory systems are still being developed and refined. The method described in this dissertation has the advantage that it requires very little knowledge of parallel programming techniques on the part of the programmer. These characteristics make the technology valuable to any number of fields.

Beyond the value of this parallelization tool is what it offers to the future of the development of protein fold recognition technologies. Over the past few CASP competitions, the best performing fold recognition techniques have been meta-servers [Ginalski and Rychlewski, 2003]. These servers poll several different fold recognitions techniques, each of them voting on the correct fold. This is because the fold recognition accuracy of a majority of the prediction techniques is within a standard deviation of each other. Each of them has a bias and works better for certain fold types and alignment pairs. By letting a group of techniques vote, minor bias and noise from each of the individual tools can be filtered out. Given this situation, a better strategy for fold recognition at the genome level would be to utilize the parallel pipeline technology to run several different fold recognition techniques

simultaneously and use the combined results to determine the correct fold. As we have already developed technology to automate parallelization, the hard parts of making the system run efficiently in a cluster environment have been resolved.

Unfortunately, most protein threading groups only provide access to their tool set through a website interface. This means that all of the calculations are rate limited by the computational resources of the lab that provides the tool. Until the tool is provided as a downloadable software package, there is no speedup gained by porting it to a high-performance computer cluster. This makes genome scale protein fold recognition less tractable, as public services provided by an independent lab will often be considerably less powerful than computational resources available at a super-computing center. There are two likely reasons for this common withholding of tools. First is the amount of effort required to document and support a publicly released threading platform. The second problem could be the culture of rivalry created by the CASP competition. Labs may feel less inclined to release their tools publicly for fear that those tools could be used against them in future CASP competitions.

The future of threading is less likely to see massive improvements in fold recognition technologies, but rather better integration of the tools involved. Protein threading is still something of a mystery to the majority of biologist and biochemists. The field has yet to achieve its 'blast' like program, i.e. that one program everyone utilizes and references. But that is a difficult goal to reach, because the best technology

in protein threading are meta-servers. To create a unified and downloadable meta-server would require a collaborative effort from several different competing labs that span the globe.

It has been clearly demonstrated by the research in this dissertation that protein threading needs to become a more integrated technology. Its connections to other protein analysis programs can be used to help inform its proper usage. This dissertation began with mining PsiBlast alignments to create better deletion models and ended with a full pipeline infrastructure to automate tool runs and integrate decisions based on the results. Protein threading is part of a larger ecology of analytical tools and the installation and utilization of all of these tools is not routine.

The most influential progress that could be offered to the field of protein structure prediction would not be in slowly pushing the accuracy of protein fold recognition ahead a few decimal points for a competition, but rather making these tools truly integrated and available to biologists at large. Right now, protein threading is a major undertaking and the web based systems are far from flexible. A new systems has to be developed such that protein threading becomes simply another way for a biologist to interactively interrogate their protein sequence.

Beyond the problem of making protein structure prediction a viable and regularly used technology, there is the question of what has been learned about the basic process of protein folding. Protein folding is still a very large problem with lots of research left to be done. The subject of fold fragments is frequently discussed when trying to explore the idea of protein folding. The idea of finding conserved

substructures in threading results taps into this concept that there basic building blocks of a protein tertiary structure. The distance matrices created by the techniques suggested in Chapter 4 could be a source of information for an investigation into these ideas.

In addition to looking at conserved substructures, my research emphasizes the negative space of the protein folding problem. By the negative space, I mean using information from similar proteins not by looking at what they have in common, but rather looking at what they do not have in common. Insertions and deletions are a theoretical construct necessary for comparing protein structure. They don't represent what the protein has, but rather what it is missing. One component may be vital to the overall structure of the protein, while another equally sized component may be completely optional. I began to explore these concepts with the idea of variable deletion energy, but this can be further pursued. The idea of dependent deletions, sections of deletions that must occur simultaneously or not at all, has still to be explored. And the algorithmic ideas explored in Chapter 5 could be adapted to aid in this research.

The models for implementing core deletions in integer programming based solutions allow for a theoretical framework for taking advantage of more complex deletion models. Currently integer programming based threading technique is not tractable for genome scale threading because of the amount of processing and memory resources it requires. But these limitations are only temporary. Computer

technology has repeatedly been driven past peoples expectations of what was possible and tractable. And while computation resources and techniques make solving such large problems difficult at the current time, it will not always be that way. And when the resources are ready, the algorithmic techniques will be available.

The biggest goal of this research is to integrate protein threading with a host of other available tools. There is still much work to be done to bring this sort of integration to an average user. And a high performance genome scale meta-server based protein threading system won't be possible until there is a massive change in the culture of protein structure prediction research.

Bibliography

Bibliography

[Alexandrov, 1996] Alexandrov, N. (1996). Sarfing the pdb. *Protein Engineering*, 9:727–732.

[Altschul et al., 1997] Altschul, S. F., Madden, T. L., Schaffer, A. A., Zhang, J., Zhang, Z., Miller, W., and Lipman, D. J. (1997). Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acids Res*, 25(17):3389–3402.

[Anfinsen, 1973] Anfinsen, C. B. (1973). Principles that govern the folding of protein chains. *Science*, 181(96):223–30.

[Bendtsen et al., 2004] Bendtsen, J. D., Nielsen, H., von Heijne, G., and Brunak, S. (2004). Improved prediction of signal peptides: SignalP 3.0. *J Mol Biol*, 340(4):783–95.

[Berman et al., 2000] Berman, H. M., Westbrook, J., Feng, Z., Gilliland, G., Bhat, T. N., Weissig, H., Shindyalov, I. N., and Bourne, P. E. (2000). The Protein Data Bank. *Nucl. Acids Res.*, 28(1):235–242.

[Bowie et al., 1991] Bowie, J. U., Luthy, R., and Eisenberg, D. (1991). A method to identify protein sequences that fold into a known three-dimensional structure. *Science*, 253(5016):164–70.

[Bradley et al., 2003] Bradley, P., Chivian, D., Meiler, J., Misura, K. M., Rohl, C. A., Schief, W. R., Wedemeyer, W. J., Schueler-Furman, O., Murphy, P., Schonbrun, J., Strauss, C. E., and Baker, D. (2003). Rosetta predictions in CASP5: successes, failures, and prospects for complete automation. *Proteins*, 53 Suppl 6:457–68.

[Bryant and Altschul, 1995] Bryant, S. H. and Altschul, S. F. (1995). Statistics of sequence-structure threading. *Curr Opin Struct Biol*, 5(2):236–44.

[Chandonia et al., 2004] Chandonia, J., Hon, G., Walker, N., Conte, L. L., Koehl, P., Levitt, M., and Brenner, S. (2004). The astral compendium in 2004. *Nucleic Acids Research*, 32:D189–D192.

[Corpet et al., 2000] Corpet, F., Servant, F., Gouzy, J., and Kahn, D. (2000). Prodom and prodom-CG: tools for protein domain analysis and whole genome comparisons. *Nucleic Acids Res*, 28(1):267–9.

[Dayhoff et al., 1978] Dayhoff, M. O., Schwartz, R. M., and Orcutt, B. C. (1978). A model for evolutionary change in proteins. *Atlas of Protein Sequence and Structure*, 5:345–352.

[Dewey., 2001] Dewey., T. G. (2001). A sequence alignment algorithm with an arbitrary gap penalty function. *Journal of Computational Biology.*, 8(2):177–190.

[Edwards and Perkins, 1996] Edwards, Y. J. and Perkins, S. J. (1996). Assessment of protein fold predictions from sequence information: the predicted alpha/beta doubly wound fold of the von willebrand factor type A domain is similar to its crystal structure. *J Mol Biol*, 260(2):277–85.

[Ellrott et al., 2006] Ellrott, K., Guo, J.-t., Olman, V., and Xu, Y. (2006). A generalized threading model using integer programming with secondary structure element deletion. *Genome Informatics*, 17(2).

[Ellrott et al., 2007] Ellrott, K., Guo, J.-t., Olman, V., and Xu, Y. (2007). Improvement in protein sequence-structure alignment using insertion/deletion frequency arrays. *Comput Syst Bioinformatics Conf. 2007*, (6):335–342.

[Fischer et al., 2003] Fischer, D., Rychlewski, L., Jr. Dunbrack, R. L., Ortiz, A. R., and Elofsson, A. (2003). CAFASP3: the third critical assessment of fully automated structure prediction methods. *Proteins*, 53 Suppl 6:503–16.

[Ginalski and Rychlewski, 2003] Ginalski, K. and Rychlewski, L. (2003). Detection of reliable and unexpected protein fold predictions using 3d-jury. *Nucleic Acids Research*, 31(13):3291–3292.

[Goonesekere and Lee, 2004] Goonesekere, N. C. and Lee, B. (2004). Frequency of gaps observed in a structurally aligned protein pair database suggests a simple

gap penalty function. *Nucleic Acids Research*, 32(9):2838–2843.

[Hardin et al., 2002] Hardin, C., Pogorelov, T. V., and Luthey-Schulten, Z. (2002). Ab initio protein structure prediction. *Curr Opin Struct Biol*, 12(2):176–81.

[Henikoff and Henikoff, 1992] Henikoff, S. and Henikoff, J. G. (1992). Amino acid substitution matrices from protein blocks. *Proc Natl Acad Sci U S A*, 89(22):10915–9.

[Ishida et al., 2003] Ishida, T., Nishimura, T., Nozaki, M., Inoue, T., Terada, T., Nakamura, S., and Shimizu, K. (2003). Development of an ab initio protein structure prediction system ABLE. *Genome Inform Ser Workshop Genome Inform*, 14:228–37.

[Jiao et al., 2006] Jiao, F., Xu, J., Yu, L., and Schuurmans, D. (2006). Protein fold recognition using the gradient boost algorithm. In *Computational Systems Bioinformatics Conference*.

[Jones, 1997] Jones, D. T. (1997). Successful ab initio prediction of the tertiary structure of NK-lysin using multiple sequences and recognized supersecondary structural motifs. *Proteins*, Suppl 1:185–91.

[Jones, 1999] Jones, D. T. (1999). Protein secondary structure prediction based on position-specific scoring matrices. *J Mol Biol*, 292(2):195–202.

[Jones et al., 1992] Jones, D. T., Taylor, W. R., and Thornton, J. M. (1992). A new approach to protein fold recognition. *Nature*, 358(6381):86–9.

[Kabsch and Sander, 1983] Kabsch, W. and Sander, C. (1983). Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 22(12):2577–637.

[Kim et al., 2003] Kim, D., Xu, D., Guo, J. T., Ellrott, K., and Xu, Y. (2003). PROSPECT II: protein structure prediction program for genome-scale applications. *Protein Eng*, 16(9):641–50.

[Kolodny et al., 2006] Kolodny, R., Petrey, D., and Honig, B. (2006). Protein structure comparison: implications for the nature of 'fold space', and structure, and function prediction. *Current Opinion in Structural Biology*, 16:393–398.

[Kuhlman et al., 2003] Kuhlman, B., Dantas, G., Ireton, G. C., Varani, G., Stoddard, B. L., and Baker, D. (2003). Design of a novel globular protein fold with atomic-level accuracy. *Science*, 302:1364–1368.

[Kurt et al., 2003] Kurt, N., Halioglu, T., and Schiffer, C. A. (2003). Structure-based prediction of potential binding and nonbinding peptides to hiv-1 protease. *Biophysical Journal.*, 85:853–863.

[Laskowski et al., 1993] Laskowski, R. A., MacArthur, M. W., Moss, D. S., and Thornton, J. M. (1993). PROCHECK: a program to check the stereochemical quality of protein structures. *J Appl Cryst*, 26:283–91.

[Lathrop, 1994] Lathrop, R. H. (1994). The protein threading problem with sequence amino acid interaction preferences is NP-complete. *Protein Eng*,

7(9):1059–68.

[Lathrop and Smith, 1994] Lathrop, R. H. and Smith, T. F. (1994). A branch and bound algorithm for optimal protein threading with pairwise (contact potential) interaction preferences. In B. Hunter, L. . S., editor, *Proc. 27th Hawaii Int. Conf. on System Sciences*, pages 365–374, Los Alamitos, CA. IEEE computer Soc. Press, Los Alamitos, CA.

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

[Lesk et al., 1986] Lesk, A. M., Levitt, M., and Chothia, C. (1986). Alignment of the amino acid sequences of distantly related proteins using variable gap penalties. *Protein Engineering*, 1(1):77–78.

[Lougee-Heimer, 2003] Lougee-Heimer, R. (2003). The common optimization interface for operations research. *IBM Journal of Research and Development*, 47(1):57–66.

[Madhusudhan et al., 2006] Madhusudhan, M., Marti-Renom, M. A., Sanchez, R., and Sali, A. (2006). Variable gap penalty for protein sequence-structure alignment. *Protein Engineering, Design, and Selection*, 19(3):129–133.

[Mizuguchi et al., 1998] Mizuguchi, K., Deane, C., Blundell, T., and Overington, J. (1998). Homstrad: a database of protein structure alignments for homologous

families. *Protein Science*, 7:2469–2471.

[Mott., 1999] Mott., R. (1999). Local sequence alignments with monotonic gap penalties. *Bioinformatics*, 15(6):455–462.

[Moult et al., 2007] Moult, J., Fidelis, K., Kryshtafovych, A., Rost, B., Hubbard, T., and Tramontano, A. (2007). Critical assessment of methods of protein structure prediction - round vii. *Proteins: Structure, Function, and Bioinformatics*, 69(S8):3–9.

[Mueller et al., 2004] Mueller, J. L., Ripoll, D. R., Aquadro, C. F., and Wolfner, M. F. (2004). Comparative structural modeling and inference of conserved protein classes in drosophila seminal fluid. *PNAS*, 101(37):13542–13547.

[Murzin et al., 1995] Murzin, A. G., Brenner, S. E., Hubbard, T., and Chothia, C. (1995). SCOP: a structural classification of proteins database for the investigation of sequences and structures. *J Mol Biol*, 247(4):536–40.

[Needleman and Wunsch, 1970] Needleman, S. B. and Wunsch, C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *J Mol Biol*, 48(3):443–53.

[Ortiz et al., 2002] Ortiz, A., Strauss, C., and Olmea, O. (2002). Mammoth (matching molecular models obtained from theory): an automated method for model comparison. *Protein Sci.*, 11(11):2606–21.

[Persson and Argos, 1994] Persson, B. and Argos, P. (1994). Prediction of transmembrane segments in proteins utilising multiple sequence alignments. *J. Mol. Biol.*, (237):182–192.

[Prijambada et al., 1996] Prijambada, D., Yomo, T., Tanaka, F., Kawama, T., Yamamoto, K., Hasegawa, A., Shima, Y., Negoro, S., and Urabe, I. (1996). Solubility of artificial proteins with random sequences. *FEBS Letters*, (383):21–25.

[Qian and Goldstein, 2001] Qian, B. and Goldstein, R. A. (2001). Distribution of indel lengths. *Proteins: Structure, Function, and Genetics*, 45:102–104.

[Reich et al., 1984] Reich, J. G., Drabsch, H., and Däumler, A. (1984). On the statistical assessment of similarities in dna sequences. *Nucleic Acids Res*, 12(13):5529–5543.

[Rohl et al., 2004] Rohl, C. A., Strauss, C. E., Misura, K. M., and Baker, D. (2004). Protein structure prediction using rosetta. *Methods Enzymol*, 383:66–93.

[Saleem et al., 2004] Saleem, R. A., Banerjee-Basu, S., Murphy, T. C., Baxevanis, A., and Walter, M. A. (2004). Essential structural and functional determinants within the forkhead domain of FOXC1. *Nucleic Acids Res*, 32(14):4182–93.

[Sali and Blundell, 1993] Sali, A. and Blundell, T. L. (1993). Comparative protein modelling by satisfaction of spatial restraints. *J Mol Biol*, 234(3):779–815.

[Shapiro and Brutlag, 2004] Shapiro, J. and Brutlag, D. (2004). FoldMiner: Structural motif discovery using an improved superposition algorithm. *Protein Sci*, 13(1):278–294.

[Shirts and Pande, 2000] Shirts, M. R. and Pande, V. S. (2000). Screen savers of the world, unite! *Science*, 290:1903–1904.

[Simons et al., 1997] Simons, K. T., Kooperberg, C., Huang, E., and Baker, D. (1997). Assembly of protein tertiary structures from fragments with similar local sequences using simulated annealing and bayesian scoring functions. *J Mol Biol*, 268(1):209–25.

[Sippl et al., 2001] Sippl, M. J., Lackner, P., Domingues, F. S., Prlic, A., Malik, R., Andreeva, A., and Wiederstein, M. (2001). Assessment of the CASP4 fold recognition category. *Proteins*, Suppl 5:55–67.

[Smith and Waterman, 1981] Smith, T. F. and Waterman, M. S. (1981). Identification of common molecular subsequences. *J Mol Biol*, 147(1):195–7.

[Vriend, 1990] Vriend, G. (1990). WHAT IF: a molecular modeling and drug design program. *J Mol Graph*, 8(1):52–6, 29.

[Wang and Dunbrack, 2003] Wang, G. and Dunbrack, J. (2003). PISCES: a protein sequence culling server. *Bioinformatics*, 19(12):1589–1591.

[Wong et al., 2001] Wong, K. K. Y., Brinkman, F. S. L., Benz, R. S., and Hancock, R. E. W. (2001). Evaluation of a structural model of pseudomonas aeruginosa

outer membrane protein oprm, an efflux component involved in intrinsic antibiotic resistance. *ournal of Bacteriology*, 183(1):367–374.

[Xiang and Honig, 2001] Xiang, Z. and Honig, B. (2001). Extending the accuracy limits of prediction for side-chain conformations. *J Mol Biol*, 311(2):421–30.

[Xu et al., 2001a] Xu, D., Baburaj, K., Peterson, C. B., and Xu, Y. (2001a). Model for the three-dimensional structure of vitronectin: predictions for the multi-domain protein from threading and docking. *Proteins*, 44(3):312–20.

[Xu et al., 2001b] Xu, D., Crawford, O. H., LoCascio, P. F., and Xu, Y. (2001b). Application of PROSPECT in CASP4: characterizing protein structures with new folds. *Proteins*, Suppl 5:140–8.

[Xu, 2005] Xu, J. (2005). Fold recognition by predicted alignment accuracy. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 2(2):157–165.

[Xu et al., 2005] Xu, J., Jiao, F., and Berger, B. (2005). A tree-decomposition approach to protein structure prediction. In *2005 IEEE Computational Systems Bioinformatics Conference*, pages 247–256, Stanford, CA, USA.

[Xu et al., 2003] Xu, J., Li, M., Kim, D., and Xu, Y. (2003). RAPTOR: optimal protein threading by linear programming. *J. Bioinform. Comput. Biol.*, 1(1):95–117.

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

[Xu et al., 2002] Xu, Y., Xu, D., and Olman, V. (2002). A practical method for interpretation of threading scores: An application of neural network. *Statistica Sinica*, 12:159–177.

[Xu et al., 1998] Xu, Y., Xu, D., and Uberbacher, E. C. (1998). An efficient computational method for globally optimal threading. *J. Comput. Biol.*, 5(3):597–614.

[Y. Guo, 2004] Y. Guo, X. (2004). PROSPECT-PSPP: An automatic computational pipeline for protein structure prediction. *Nucl. Acids. Res.*, 32:W522–5.

[Yona and Levitt, 2002] Yona, G. and Levitt, M. (2002). Within the twilight zone: a sensitive profile-profile comparison tool based on information theory. *J Mol Biol*, 315(5):1257–75.

[Zhang and Kim, 2000] Zhang, C. and Kim, S.-H. (2000). Environment-dependent residue contact energies for proteins. *Proceedings of the National Academy of Sciences*, 97(6):2550–2555.

[Zhang et al., 2004a] Zhang, C., Liu, S., Zhou, H., and Zhou, Y. (2004a). An accurate, residue-level, pair potential of mean force for folding and binding based on the distance-scaled, ideal-gas reference state. *Protein Sci.*, 13:400–411.

[Zhang et al., 2004b] Zhang, C., Liu, S., and Zhou, Y. (2004b). Accurate and efficient loop selections by the DFIRE-based all-atom statistical potential. *Protein Sci*, 13(2):391–9.

[Zhu and Weng, 2005] Zhu, J. and Weng, Z. (2005). FAST: A novel protein structure alignment algorithm. *Proteins*, 58:618–627.

[Zien et al., 2000] Zien, A., Zimmer, R., and Lengauer, T. (2000). A simple iterative approach to parameter optimization. In *Recomb Proceedings 2000*, pages 318–327.

Appendix

Appendix

Mandelbrot Test XML description

```
<config>
   <source name="mandel">mandelbrot.py</source>
   <pipeline name="mandle_1_split">
‘       <input name="res"/>
      <output name="data">print_out:confirm</output>
      <node name="split_x">
         <func>mandel.x_range</func>
         <input name="res">res</input>
         <output name="x_array" special="return_val" />
      </node>
      <node name="calc_y_array">
         <func>mandel.y_array_calc</func>
         <input name="x_val" cmd="array_split">split_x:x_array</input>
         <input name="res" cmd="replicate">res</input>
         <output name="out_array" cmd="array_collect" special="return_val"/>
      </node>
      <node name="print_out">
         <func>mandel.print_xy_list</func>
         <input name="xy_array">calc_y_array:out_array</input>
         <input name="res">res</input>
         <output name="confirm" special="return_val"/>
      </node>
   </pipeline>
</config>
```

Mandelbrot Test Python Script

```
maxiteration = 1000

def calc_pixel( x, y ):
   cur_x = x
   cur_y = y
   iteration = 0
   while ( cur_x*cur_x + cur_y*cur_y < 4 and  iteration < maxiteration ):
      xtemp = cur_x*cur_x - cur_y*cur_y + x
```

```
      cur_y = 2*cur_x*cur_y + y
      cur_x = xtemp
      iteration = iteration + 1
   return iteration

def x_range(res):
   x_out = []
   for a in range( -(int(res)/2), (int(res)/2) ):
      x_out.append( (float(a)/int(res))*2 - .5)
   return x_out

def y_array_calc( x_val, res ):
   y_calc_val = []
   for b in y_range(int(res)):
      y_calc_val.append( calc_pixel( x_val, b ) )
   return y_calc_val

def print_xy_list( xy_array, res ):
   print res, res
   for a in xy_array:
      for b in a:
         print b
   return ""
```

Vita

Kyle Ellrott was born in Portland, Oregon, on December 28th 1978, the son of Phil and Kim Ellrott. Before graduating from North Medford High in Medford Oregon, Kyle had lived and gone to school in Brazil and Portugal. He attended University of California, Riverside where he received a Bachelors of Science degree from the Computer Science department in 2002. During his studies, he began an internship at Oak Ridge National Labs in Oak Ridge Tennessee, which continued until he accepted into the Genomic Science and Technology program in the Life Science Department of University of Tennessee, Knoxville. In 2003, he moved to University of Georgia, in Athens Georgia, to follow his major professor Dr. Ying Xu. There he continued to work on completing his Doctorate degree from the University of Tennessee, which he received in 2007.