Computer Sciences Department

Probabilistic Methods for Interpreting Electron-Density Maps

Frank Paul DiMaio

Technical Report #1617

October 2007

PROBABILISTIC METHODS FOR INTERPRETING ELECTRON-DENSITY MAPS

by

Frank Paul DiMaio

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctor of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2007

ACKNOWLEDGMENTS

I am fortunate to have been assisted by many people during my graduate career.

I first would like to acknowledge my advisor, Jude Shavlik, who has been a wonderful teacher and mentor. He has played a very important role in my graduate career, and a great majority of what I learned in graduate school is due to him.

I would also like to thank the members of my committee, all of whom have been great sources of guidance: George Phillips, Mark Craven, David Page, and Chuck Dyer. I learned an incredible amount of biology from conversations with George. Mark and David, as part of the machine learning group, have both had tremendous influence on my graduate career. What was to become this thesis began as a class project in Chuck's class.

I would also like to thank my collaborators, Ameet Soni, Dmitry Kondrashov, Ed Bitto and Craig Bingman. Many important ideas in this thesis began as conversations with Ameet and Dmitry.

I would like to thank all of the people who were fellow students in the machine learning group at one time or another during my graduate career: Soumya Ray, Joe Bockhorst, Aaron Darling, Sean McIlwain, Yue Pan, Adam Smith, Mark Goadrich, Burr Settles, Michael Waddell, Mike Molla, Keith Noto, Louis Oliphant, Trevor Walker, Lisa Torrey, Jesse Davis, Beverly Seavey, Irene Ong, Dave Andrzejewski and Eric Lantz.

Finally, I want to thank my sources of funding, the University of Wisconsin Graduate School and the National Library of Medicine (1R01 LM008796 and 1T15 LM007359). I would also like to acknowledge the Computation and Informatics in Biology and Medicine Training Program and Louise Pape.

TABLE OF CONTENTS

Page

LIST OF TABLES vi

LIST OF FIGURES vii

LIST OF ALGORITHMS xi

ABSTRACT xii

1 Introduction 1

1.1 Electron-density map interpretation 1
1.2 Probabilistic reasoning 3
1.3 Thesis statement 4
1.4 Outline 4

2 Background and Related Work 6

2.1 Background 6
2.1.1 Protein structure 7
2.1.2 X-ray crystallography 9
2.1.3 Algorithmic background 13
2.1.4 Undirected graphical models 15
2.2 Alternative approaches to automatic density-map interpretation 18
2.2.1 ARP/wARP 20
2.2.2 RESOLVE 25
2.2.3 TEXTAL 32
2.2.4 Summary of approaches 37
2.3 Discussion 38

3 A Probabilistic Approach to Protein-Backbone Tracing 39

3.1 Overview of the algorithm 39
3.2 Local matching 41

Page

3.2.1 Constructing a sequence-specific 5-mer basis set . . . 41
3.2.2 Searching the map for 5-mer template structures . . . 42
3.2.3 Additional sources of local information . . . 45
3.2.4 Discussion . . . 46
3.3 Global constraints . . . 46
3.3.1 Pairwise Markov-field model . . . 48
3.3.2 ACMI-BP's inference algorithm . . . 52
3.3.3 Technical challenges . . . 53
3.4 Experiments . . . 55
3.5 Conclusions and future work . . . 57

4 Improved Template Matching Using Spherical Harmonics . . . 60

4.1 The goal of template matching . . . 60
4.2 Spherical harmonics and the fast rotation function . . . 61
4.3 A method for fast template matching . . . 63
4.3.1 Overview of the approach . . . 63
4.3.2 Advantages of rotational "convolution" . . . 66
4.3.3 Modified pentapeptide templates . . . 67
4.4 Results . . . 68
4.4.1 Errors in band-limiting density . . . 69
4.4.2 First-pass filtering . . . 70
4.4.3 Template matching . . . 71
4.4.4 Comparison of protein models produced . . . 72
4.5 Conclusions and future work . . . 72

5 Creating All-Atom Protein Models using Particle Filtering . . . 75

5.1 Shortcomings in ACMI-BP's model . . . 75
5.2 Particle-filtering overview . . . 77
5.3 ACMI-PF's protein-particle model . . . 78
5.3.1 Using ACMI-BP-computed marginals to place Cα's . . . 79
5.3.2 Using sidechain templates to sample sidechains . . . 83
5.3.3 Sampling order . . . 85
5.4 Experiments . . . 86
5.4.1 Methodology . . . 86
5.4.2 ACMI-NAÏVE versus ACMI-PF . . . 87
5.4.3 Comparison to other algorithms . . . 89
5.5 Iterative phase improvement with ACMI-PF . . . 91
5.5.1 The phase problem . . . 91

Page

5.5.2 Using ACMI-PF for phase improvement . . . 92
5.5.3 Multiple iterations of ACMI-PF . . . 93
5.6 Conclusions and future work . . . 94

6 Pictorial Structures for Atom-level Models . . . 97

6.1 Pictorial structures . . . 97
6.2 Building a flexible atomic model . . . 98
6.3 Model enhancements . . . 102
6.3.1 Collision detection . . . 103
6.3.2 Improved template matching . . . 103
6.4 Experimental studies . . . 105
6.5 Conclusions and future work . . . 105

7 Improving the Efficiency of Belief Propagation . . . 107

7.1 Introduction . . . 107
7.2 Modeling 3D objects . . . 108
7.3 Scaling belief propagation . . . 110
7.3.1 BP message aggregation . . . 110
7.3.2 Message representation . . . 112
7.4 Experiments . . . 116
7.4.1 Protein-fragment identification . . . 116
7.4.2 Synthetic-object recognition . . . 118
7.5 Conclusions and future work . . . 124

8 Conclusion . . . 125

8.1 Probabilistic protein-backbone tracing . . . 126
8.2 Improved template matching in density maps . . . 126
8.3 Constructing protein models using particle filtering . . . 127
8.4 Atom-level matching using pictorial structures . . . 128
8.5 A general object-recognition framework . . . 128
8.6 Final wrapup . . . 128

APPENDICES

Appendix A: Datasets . . . 130
Appendix B: Supplementary experiments . . . 134

GLOSSARY . . . 137

Page

LIST OF REFERENCES 139

LIST OF TABLES

Table Page

2.1 The rotation-invariant features used by TEXTAL 34

5.1 The use of multiple protein structures reduces phase errors more than a single structure 94

A.1 The four proteins in Dataset 1 130

A.2 The ten proteins in Dataset 2 131

A.3 The ten proteins in Dataset 3 133

LIST OF FIGURES

Figure Page

1.1 An overview of density-map interpretation 2

2.1 Proteins are constructed by joining chains of amino acids in peptide bonds 7

2.2 A protein's primary, secondary, and tertiary structure 8

2.3 An overview of the crystallographic process 9

2.4 An all-atom protein model and corresponding Cα trace 10

2.5 A tryptophan residue's density at several resolutions 11

2.6 A tryptophan residue's density as mean phase error varies 12

2.7 A simple pairwise Markov-field model 16

2.8 The 2.5Å resolution electron-density map of the 95-amino-acid protein 1XMT 21

2.9 A flowchart of ARP/wARP's WARPNTRACE 22

2.10 Intermediate steps in ARP/wARP's structure determination 23

2.11 A flowchart of RESOLVE 25

2.12 The averaged helix and strand fragment used in RESOLVE's initial matching step . . . 26

2.13 Intermediate steps in RESOLVE's structure determination 27

2.14 A flowchart of TEXTAL 32

2.15 Intermediate steps in TEXTAL's structure determination 35

3.1 The 5-mer clustering process 42

3.2 An overview of the 5-mer template matching process 43

Figure Page

3.3 Backbone traces maximizing prior probabilities and posterior probabilities 47

3.4 The structure of ACMI-BP's graphical model 48

3.5 An illustration of an amino acid's internal rotational parameters 50

3.6 The angular constraint function p_Θ 51

3.7 A simple example of message passing using belief propagation 53

3.8 An illustration of occupancy message aggregation 55

3.9 Graphs comparing ACMI's, TEXTAL's, and RESOLVE's interpretation 57

3.10 A scatterplot showing ACMI's performance on a protein-by-protein basis 58

3.11 An illustration of predicted versus actual structure on ACMI's sixth-best prediction (out of ten) at 3.5Å resolution 58

4.1 The real and imaginary components of several low-order spherical harmonics 62

4.2 An overview of spherical-harmonic decomposition 63

4.3 ACMI-SH's improved template-matching algorithm 64

4.4 Differences between ACMI-FF's and ACMI-SH's density templates 68

4.5 The average squared density difference between a region of sampled density and the bandwidth-limited region 69

4.6 A comparison of four different filters for quickly eliminating some portion of the density map 70

4.7 A comparison of ACMI-SH's and ACMI-FF's template matching 71

4.8 A comparison of ACMI-SH+BP's protein models with three other methods 73

5.1 One case where a maximum-marginal trace may be undesirable 76

5.2 A pictorial look at particle filtering 78

5.3 Conditional dependencies in sidechain and Cα layout 79

Figure Page

5.4 An overview of the backbone forward-sampling step 82

5.5 An overview of the sidechain sampling step 84

5.6 A comparison of the R_{free} of ACMI-NAÏVE and ACMI-PF 88

5.7 A comparison of ACMI-PF's model completeness to three other automatic interpretation methods 89

5.8 A comparison of ACMI-PF's free R factor to three other automatic interpretation methods 90

5.9 An overview of iterative phase improvement in crystallographic density maps 92

5.10 A scatterplot comparing the error of ACMI-PF's calculated phases with the error of the experimentally estimated phases 93

5.11 Scatterplots comparing results after one and two iterations of ACMI 95

5.12 Mean phase error as a function of iteration 95

6.1 DEFT's construction of the pictorial-structure graph given an amino acid 99

6.2 The screw-joint, which connects atoms in DEFT's model 99

6.3 DEFT's parameter-learning process 102

6.4 An example of DEFT's collision correction 103

6.5 DEFT's leave-one-protein-out testset errors 106

7.1 A graphical model for recognizing a person in an image 110

7.2 AGGBP's message aggregation 113

7.3 A comparison of memory and CPU time usage between AGGBP and LOOPYBP . . . 116

7.4 A comparison of AGGBP to LOOPYBP at each iteration of message passing 117

7.5 RMS error of AGGBP and LOOPYBP as a function of protein fragment size 117

Figure Page

7.6 A scatterplot comparing log likelihoods of AGGBP's and LOOPYBP's maximum-marginal backbone models 119

7.7 Four graph-topology parameters that one may vary using my graph generator 120

7.8 Observation potentials are generated by drawing scores from two distributions 120

7.9 A comparison of AGGBP and LOOPYBP using the synthetic-object generator 122

A.1 Data was downsampled by smoothly diminishing reflection intensities 131

A.2 A scatterplot showing experimental data quality in terms of the map resolution and mean phase error 132

B.1 A comparison of a single template's contribution to the observation potential ψ_i to the mixture-of-Gaussian approximation 135

B.2 A comparison of ACMI's observation potential ψ_i to the mixture-of-Gaussian approximation 136

B.3 The KL-divergence of AGGBP's messages versus LOOPYBP's messages as a function of Cα–Cα distance 136

LIST OF ALGORITHMS

Algorithm Page

2.1 Belief propagation 17

2.2 WARPNTRACE's model-building algorithm 24

2.3 RESOLVE's chain-assembly algorithm 29

2.4 RESOLVE's sidechain-placement algorithm 31

2.5 TEXTAL's CAPRA subroutine for calculating the initial backbone trace 34

2.6 TEXTAL's LOOKUP subroutine for placing sidechains 36

3.1 ACMI's algorithm for inferring protein Cα locations 40

4.1 ACMI-SH's template matching 65

5.1 ACMI-PF grows a protein model in two phases 80

5.2 ACMI-PF generates multiple models through multiple independent runs 86

6.1 DEFT's collision-handing routine 104

7.1 Aggregate belief propagation 114

ABSTRACT

With recent advances in structural genomics, there has been considerable interest in the rapid determination of protein structures in a high-throughput setting. One bottleneck in this process arises in protein crystallography, and deals with interpretation of the electron-density map, the three-dimensional "picture" of the protein that crystallography produces. This thesis presents a novel solution to this important problem of electron-density map interpretation. I apply probabilistic methods to automate the interpretation of poor-quality electron-density maps.

I show my probabilistic approach to density-map interpretation leads to more complete and more accurate protein models, in terms of the fraction of the protein automatically interpreted, as well as the RMS error of my method's inferred models versus "ground truth" (the deposited structure), than do other automated approaches. My probabilistic approach is also amenable to production of multiple protein models that explain the observed density. I show that multiple static conformations generated by my framework do a better job of explaining the observed density than does a single structure, based on the R_{free} metric, which measures the difference between observed and predicted crystallographic reflection data on a testset of held-aside data. My method accurately interprets 3-4Å density maps, further extending the resolution of density maps that can be automatically interpreted.

This thesis also describes several computational contributions. I describe a significant improvement over previous work in three-dimensional template matching in electron-density maps. I use the spherical-harmonic decomposition of a template to rapidly search for all rotations of the template. This offers both improved efficiency and accuracy compared to previous work, producing better models in 60% of the running time. I present a novel joint type as well as improved methods for collision-handling in part-based object-recognition. Finally, I present a general part-based object-recognition framework specialized for identifying topologically complex objects in large three-dimensional images. My framework introduces an algorithm that improves the efficiency of current probabilistic inference algorithms. This improved efficiency allows recognition of objects with hundreds of parts. Although originally developed for density-map interpretation, these computational contributions may be beneficial in other problem domains.

Chapter 1

Introduction

In recent years, considerable investment into high-throughput determination of protein structures has yielded a wealth of new data. The demand for rapid structure solution is growing, and automated methods are being deployed at all stages of the structure determination process. New technologies includes cell-free methods for protein production [100], the use of robotics to enable massive arrays of crystallization conditions [104], and software for automated building of macromolecular models based on the electron-density map [27, 55, 84, 110].

My thesis concerns this last problem: given an *electron-density map* – a three-dimensional image of a protein – one wants to identify the location of each atom in the protein in this image. Traditionally, a human performs this *interpretation*, perhaps aided by a graphics display. Recently, however, algorithms for automatically constructing such a model have come into general use. If the image quality is sufficiently good, these algorithms produce quite accurate models with little human intervention. However, when the image quality is poor, significant human input is required. Methods for accurate model construction in noisy, poor-quality maps are important to high-throughput protein-structure determination.

Additionally, rich three-dimensional data commonly arises in other biological domains, particularly with recent advancements in biological-imaging techniques. For example, fMRI scans [102] produce detailed 3D images of the brain. Confocal microscopy [92] constructs high-quality 3D images of tissues. Electron cryomicroscopy (cryo-EM) [12] gives detailed images of large macromolecular complexes. This three-dimensional data often contains objects comprised of many parts, connected with some complex topology.

With easier acquisition of detailed biological imagery, techniques to accurately interpret such images are needed. A vascular biologist may want to automatically locate all the blood vessels in a kidney section. A virologist may want to identify the 3D geometry of a virus capsid from a cryo-EM scan. Even rich two-dimensional data, such as detailed satellite imagery, may contain complex objects that cannot be fully interpreted using current methods. As with density-map interpretation, methods for automatically analyzing this complex, three-dimensional data are critically important.

1.1 Electron-density map interpretation

The electron-density map is a real-valued function defined on a three-dimensional grid of points covering the *unit cell*, which is the basic repeating unit in the protein crystal. A crystallographer

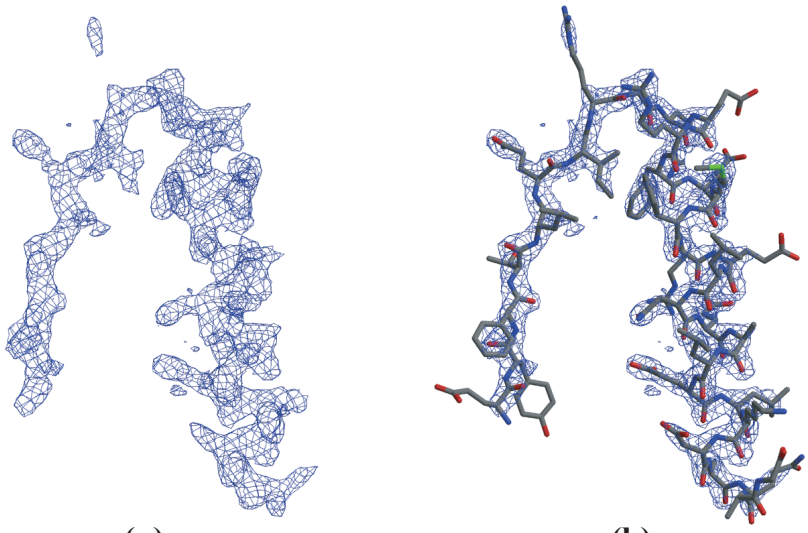


Figure 1.1 An overview of density-map interpretation. (a) A raw density map, with contours enclosing regions of higher density, and (b) with a model of predicted protein structure, where sticks indicate bonds between atoms.

starts with the amino-acid sequence of the protein, and attempts to place these amino acids in the unit cell, based on the shape of the electron-density contours.

My thesis addresses the use of probabilistic reasoning to automatically produce an all-atom protein model from an electron-density map. The general task I address is:

Given:	electron-density map M amino-acid sequence seq
Predict:	3D coordinates (x, y, z) of each atom in the protein

The task is depicted graphically in Figure 1.1. Given the electron-density map – illustrated as an isocontoured surface – in Figure 1.1a, I want to *automatically* produce the model illustrated in Figure 1.1b, that is, a model of atomic positions consistent with the electron-density (sticks indicate bonds between atoms). This task is *biologically* challenging for several reasons:

- The difficulty in coercing certain proteins to crystallize often leads to the production of small, poor-quality crystals. The data one can collect from such crystals is limited, leading to poor density-map *resolution*. In these poor-resolution maps, individual atoms are no longer visible.

- Due to *phasing error*, density maps often suffer from significant noise. Before a map is computed, a crystallographer must measure reflection intensity and estimate reflection phases.

Phasing error is due to the fact that experimentally determined phases are typically inaccurate [97], leading to distortion in the density map [15]. Noise in data collection may also introduce distortion into the density map. Identifying and isolating a single copy, while combining information from multiple copies is challenging from a computational perspective.

- An electron-density map is really an "average" over a 3D matrix of protein molecules in the crystal. *Flexible regions* in the protein are averaged out in this image, producing portions of the protein chain for which little or no density is visible. Crystallographers can use their structural "background knowledge" to fill in gaps, but in many maps, large portions of the protein molecule are left unstructured, even in the final model deposited in the Protein Data Bank [3].

Throughout the remainder of my thesis, I show that a probabilistic approach to the problem of density-map interpretation is able to address each of these difficulties. By building a model based on the *amino-acid sequence* of a protein, and finding the most-probable position of each atom *given* the sequence and the density map, I am able to more accurately interpret poor-quality maps.

1.2 Probabilistic reasoning

Probabilistic models [6, 60], such as Bayesian networks and Markov fields, provide a natural way of representing uncertainty in the outcome of an event. These models are particularly well-suited to handling *noisy data*, *hidden (i.e., unobserved) states*, and *complex interactions* between variables. They have been successfully applied to a number of biological problems [51], where these types of complexities are common. A key reason for their success is their expressive power.

Probabilistic reasoning, or *inference*, tries to say something about some variable in a probabilistic model, given some observations. While probabilistic models are expressively powerful, inference becomes more difficult as model complexity increases. One key challenge in machine learning is inference in complex probabilistic models [48].

Density-map interpretation is *computationally* challenging for several reasons:

- There is significant local redundancy in the data. A particular amino acid may appear in dozens of places in the density map; with the exception of several large, ring-containing amino acids, many amino acids appear quite similar. To align the sequence to the map requires one take global view of the protein.

- Proteins are quite large – often containing several thousand amino acids – and flexible. No two atoms can occupy the same 3D location, so one cannot easily employ a divide-and-conquer strategy, placing portions of the protein independently.

- Density maps are quite large. Accurately modeling probability distributions over a density map requires one make significant simplifying assumptions.

My thesis proposes several novel machine-learning contributions to deal with these computational challenges. A key contribution is the idea of aggregating amino-acids in the protein, and approximating their combined probability distribution with a single probability distribution.

The inference techniques I develop also address the more general problem of identifying complex, highly flexible objects from images. I develop a general framework for identifying part-based models in image data. I extend the idea of "part aggregation" to generic part-based object models, enabling tractable inference for complex objects with many flexible parts.

1.3 Thesis statement

I hypothesize that a probabilistic approach to density-map interpretation will lead to *more complete* protein models, in terms of fraction of the protein placed in the density map, and *more accurate* protein models, in terms of sidechain identification and RMS deviation versus "ground truth" (the structure deposited in the protein data bank [3]), than other automated approaches. I propose a probabilistic method to automate the interpretation of poor-quality electron-density maps. A probabilistic model known as a pairwise Markov random field represents amino acids of the protein as nodes in a graph, while spatial constraints between pairs of amino-acids are modeled as edges. My probabilistic approach is also amenable to production of multiple protein models that explain the observed density. I hypothesize that multiple static conformations do a better job explaining the observed density than does a single structure, based on the R_{free} metric, which measures the difference between observed and predicted reflections on a testset of held-aside reflections.

1.4 Outline

The remainder of my thesis is organized in the following manner:

- Chapter 2 contains relevant background material. I first present essential computational and probabilistic material, followed by a detailed description of density-map interpretation. In this chapter, I also present in detail three alternate approaches (of others) from the literature, illustrating how each algorithm approaches this task.

- Chapter 3 describes a probabilistic model for inferring a coarse protein structural model. I describe construction of my probabilistic model – based on a protein's amino-acid sequence – that assigns a probability to each potential structure, and how I infer the most-probable structure under such a model.

- Chapter 4 investigates further one particular subtask of the backbone-inference problem: how to rapidly search the entire map for a small density template. I describe the use of spherical harmonics to rapidly search a density map for all rotations of a template. I show that using a spherical harmonic decomposition to search for small templates locates template instances more accurately and in less time than previous approaches.

- Chapter 5 describes how an all-atom protein model is constructed from the inferred backbone using particle-filtering methods. I consider growing an ensemble of protein models one amino acid at a time, resampling the ensemble at regular intervals to maintain a set of high-probability partial models. I empirically show that the resultant models are more accurate than those produced by other methods.

- Chapter 6 explores at an alternate approach to constructing an all-atom model, using a part-based model similar to the backbone model of Chapter 3. I describe a new joint type for modelling covalently bonded atoms. An efficient dynamic programming-algorithm finds the most likely placement of individual atoms in the model. Several variations improve the quality of atom placement.

- Chapter 7 generalizes the framework introduced Chapter 3, describing a generic part-based object-recognition model. I also introduce the idea of "message aggregation" for belief propagation, which makes inference tractable in objects – like proteins – with many flexible connections.

- Chapter 8 concludes my thesis, describing contributions, lessons learned, and summarizes some interesting possible directions for future research.

- Two appendices present additional material outside the scope of the thesis: Appendix A describes the various datasets I used for testing, while Appendix B presents some supplementary experimental data. A glossary defines frequently used terminology.

Chapter 2

Background and Related Work

The purpose of this chapter is to present a brief background of biological and computational material relevant to the remainder of this thesis. This chapter begins with a brief introduction to structural biology and X-ray crystallography. I describe the problem of electron-density map interpretation in detail, and introduce several other algorithms used for automatic interpretation. In this chapter, I also describe three widely used methods in greater detail. I apply each algorithm to a sample density map, illustrating each algorithm's intermediate steps and the resulting interpretation. For each algorithm, I present pseudocode and program-flow diagrams, and discuss advantages and shortcomings.

2.1 Background

Knowledge of a protein's *fold* – that is, the sequence-determined, three-dimensional structure – is valuable to biologists. A protein's structure provides great insight into the mechanisms by which a protein acts, and knowing these mechanisms helps increase our basic understanding of the underlying biology. Structural knowledge is increasingly important in disease treatment, and has led to the creation of catalysts with industrial uses. No existing computer algorithm can accurately map sequence to 3D structure; however, several experimental (i.e., "wet lab") techniques exist for determining macromolecular structure. The most commonly used method, employed for about 80% of structures currently known, is *X-ray crystallography*. This time-consuming and resource-intensive process uses the pattern of X-rays diffracted off of a crystallized matrix of protein molecules to produce an electron-density map. This electron-density map is a three-dimensional "picture" of the electron clouds surrounding each protein atom. Producing a protein structure is then a matter of identifying the location of each of the protein's atoms in this 3D picture.

Density map interpretation – traditionally performed by a crystallographer – is time-consuming and, in noisy or poor-quality maps, often error-prone. Recently, a number of research groups have looked into automatically interpreting electron-density maps, using ideas from machine learning and computer vision. These methods have played a significant role in high-throughput structure determination, allowing novel protein structures to more quickly be elucidated.

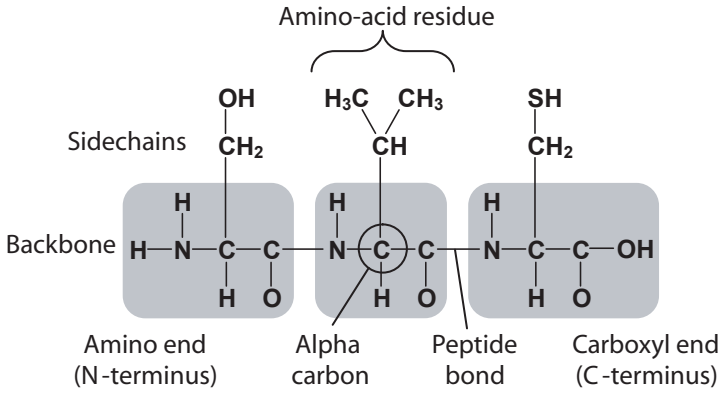


Figure 2.1 Proteins are constructed by joining chains of amino acids in peptide bonds. A chain of three amino-acid residues is illustrated.

2.1.1 Protein structure

Proteins (also called *polypeptides*) are constructed from a set of building blocks called amino acids. Each of the twenty naturally-occurring amino acids consists of an amino group and a carboxylic acid group on one end, and a variable chain on the other. When forming a protein, adjacent amino groups and carboxylic acid groups condense to form a repeating backbone (or mainchain), while the variable regions become dangling sidechains. Each condensed amino acid in the protein is referred to as an *amino-acid residue*, or simply a *residue* for short. The atom at the interface between the sidechain and the backbone is known as the alpha carbon, or Cα (see Figure 2.1). The linear list of amino acids composing a protein is often referred to as the protein's primary structure (see Figure 2.2a).

A protein's secondary structure (see Figure 2.2b) refers to the common three-dimensional structural motifs taken by continuous segments in the protein. There are two such motifs: α-helices, in which the peptide chain folds in a corkscrew, and β-strands, where the chain stretches out linearly. In most proteins, several β-strands run parallel or antiparallel to one another. These regular structural motifs are connected by less-regular structures, called loops (or turns). A protein's secondary structure can be predicted somewhat accurately from its amino-acid sequence [98].

Finally, a protein's three-dimensional conformation – also called its tertiary structure (see Figure 2.2c) – is uniquely determined from its amino-acid sequence (with some exceptions). No existing computer algorithm can accurately map sequence to tertiary structure; instead, one must rely on experimental techniques, primarily X-ray crystallography, to determine tertiary structure.

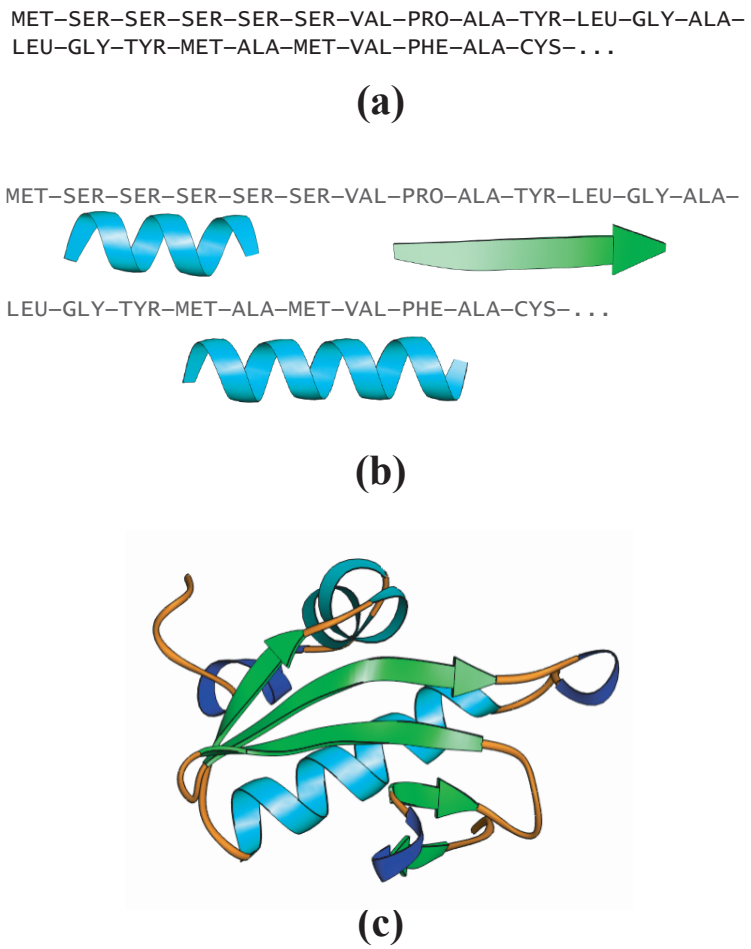


Figure 2.2 An illustration of (a) a protein's *primary structure*, the linear amino-acid sequence of the protein, (b) a protein's *secondary structure*, which describes local structural motifs such as alpha helices and beta sheets, and (c) a protein's *tertiary structure*, the global three-dimensional conformation of the protein.

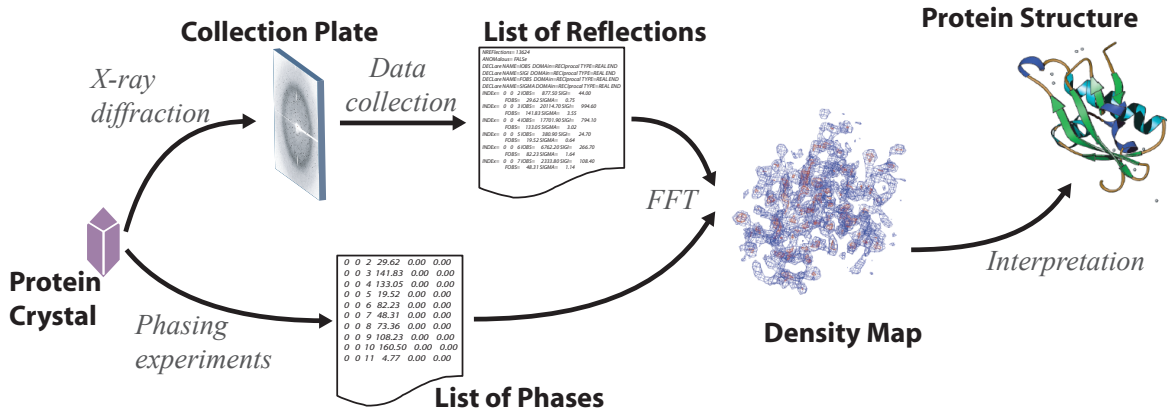


Figure 2.3 An overview of the crystallographic process.

2.1.2 X-ray crystallography

An overview of protein crystallography appears in Figure 2.3. Given a suitable target for structure determination, a crystallographer must produce and purify this protein in significant quantities. Then, for this particular protein, one must find a very specific setting of conditions (i.e., pH, solvent type, solvent concentration) under which protein crystals will form. Once a satisfactory crystal forms, it is placed in front of an X-ray source. Here, this crystal diffracts a beam of X-rays, producing a pattern of spots on a collector. These spots – also known as *reflections* or *structure factors* – represent the Fourier-transformed electron-density map. Unfortunately, the experiment can only measure the intensities of these (complex-valued) structure factors; the phases are lost.

Determining these missing phases is known as the *phase problem* in crystallography, and can be solved to a reasonable approximation using computational or experimental techniques [97]. Only after estimating the phases can one compute the electron-density map .

The *electron-density map* (which I will refer to as a *density map* or simply *map* for short) is defined on a 3D lattice of points covering the *unit cell*, the basic repeating unit in the protein crystal. The crystal's unit cell may contain multiple copies of the protein related by crystallographic symmetry, one of the 65 regular ways a protein can pack into the unit cell. Rotation/translation operators relate one region in the unit cell (the asymmetric unit) to all other symmetric copies. Additionally, the protein may form a multimeric complex (e.g. a dimer, tetramer, etc.) within the asymmetric unit. In all these cases it is up to the crystallographer to isolate and interpret a single copy of the protein.

Figure 1.1 (in the previous chapter) shows a sample fragment from an electron-density map, and the corresponding interpretation of that fragment. The amino-acid (primary) sequence of the protein is typically known by the crystallographer before interpreting the map. In a high-quality map, every single (non-hydrogen) atom in the protein can be placed in the map.

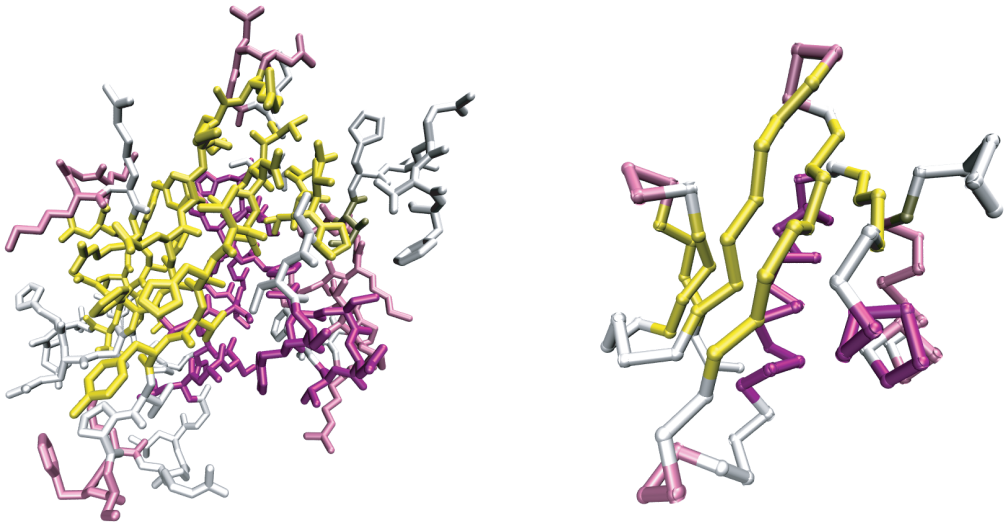


Figure 2.4 (a) An all-atom protein model and (b) the corresponding *backbone trace*, which represents each residue by its Cα location.

However, in a poor-quality map it may only be possible to determine the location of a single key atom, the alpha carbon or Cα, in each residue. This is known as a *backbone trace* or a *Cα trace*. Figure 2.4 shows an all-atom protein model and the corresponding backbone trace. A backbone trace – though not as comprehensive as an all-atom model – is still valuable to biologists.

The quality of an electron-density map is limited by its *resolution*, which, at its high limit, corresponds to the smallest interplanar distance between diffracting planes. The highest resolution for a data set depends on the order in the crystalline packing, the detector sensitivity, and the brightness of the X-ray source. Figure 2.5 illustrates the electron density around a tryptophan sidechain at varying resolution, with "ideal" phases computed from a complete all-atom model. Note that at 1 Å resolution, the spheres of individual atoms are clearly visible, while at 4 Å even the overall shape of the tryptophan sidechain is distorted. Typical resolution for protein structures lies in the 1.5 – 2.5 Å range.

Another factor that affects the quality of an electron-density map is the accuracy of the computed phases. To obtain an initial approximation of the phases, crystallographers use techniques based on the special features in X-ray scattering produced by heavy atoms, such as multiple-wavelength [44] or single-wavelength anomalous diffraction [46] (MAD or SAD) and multiple isomorphous replacement (MIR) [8]. This allows the computation of an initial electron-density map, the quality of which greatly depends on the fidelity of the initial phasing. Artifacts produced by phase error are similar to those of worsening resolution; additionally, high spatial frequency noise is also present.

As the model is built, these phases are iteratively improved [1], producing a better quality map, which may require resolving large portions of the map. Figure 2.6 illustrates the effect poor

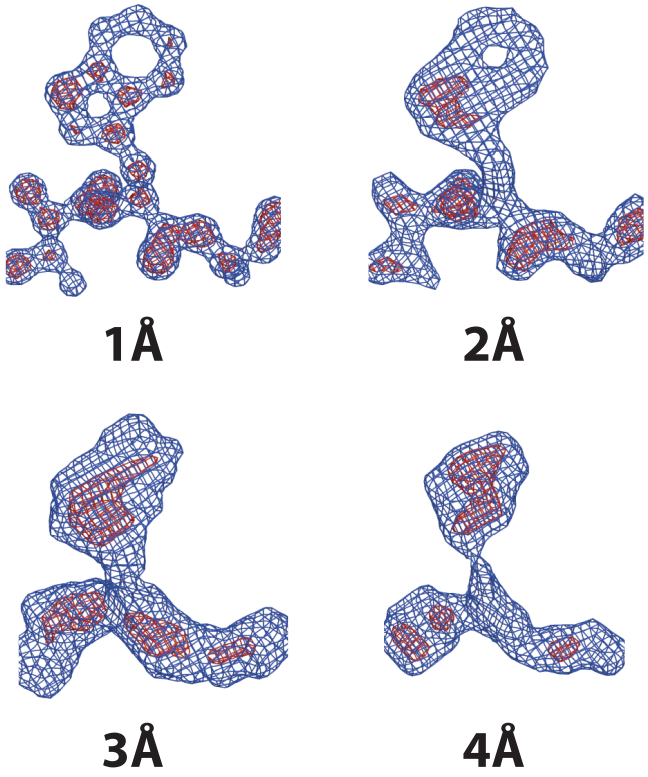


Figure 2.5 A tryptophan residue's density at several resolutions.

phasing has on density-map quality. This figure was generated by adding Gaussian noise to each reflection phase, and recomputing the density map. In addition, noise in diffraction-pattern data collection also introduces errors into the resulting density map.

Finally, the density map produced from X-ray crystallography is not an image of a single molecule, but rather an average over an ensemble of all the molecules contained within a single crystal. Flexible regions in the protein are not visible at all, because they are averaged out.

For most proteins, this interpretation is done by an experienced crystallographer, who can, with high-quality data, fit about 100 residues per day in an interactive computer graphics environment [59, 79]. However, for poor-quality density maps, interpretation can be an order of magnitude slower. The interpretation of a poorly phased map can be very difficult even for a trained expert.

Given some putative atomic structure, one can compute the model-determined (or calculated) structure factors F_{calc} as the scattering one would *expect* to see. It is calculated by summing the contribution of each individual atom in the model to the overall X-ray scattering. Typically, each atom is modeled as a single sphere of electron density; each atom's scattering factor is modeled with a single Gaussian (or four Gaussians with high-resolution maps) [19]. These Gaussians are different for each atom, and are based on a fit to experimental scattering data.

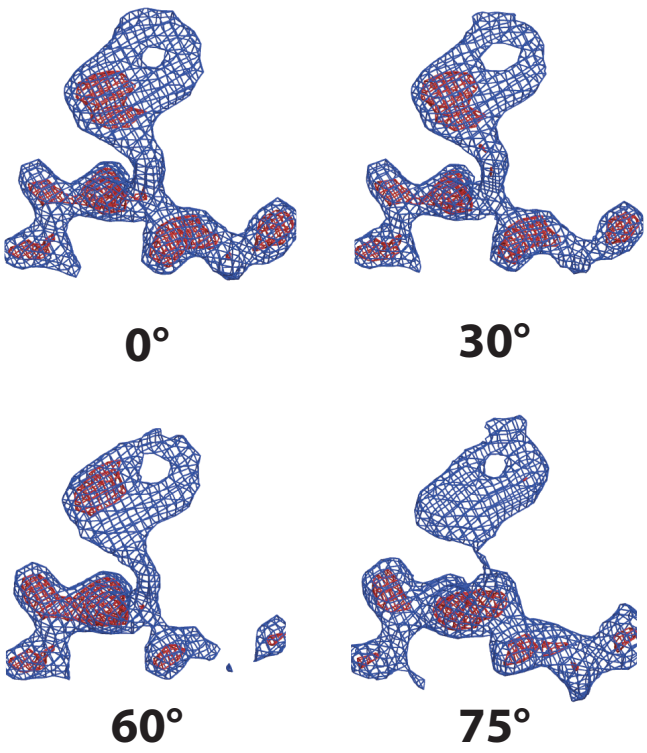


Figure 2.6 A tryptophan residue's density as mean phase error varies.

This allows one to compute a *model-determined* (or *calculated*) *density map*: the electron density one would expect to see, given some atomic structure. Calculated density maps are used for training by many automated methods (including my own) since many more protein structures are available than experimental density maps [3].

A key question for computational methods for interpreting density maps is the following: how are candidate 3D models scored? Crystallographers typically use a model's R *factor* (for *residual index*) to evaluate the quality of an interpretation. Formally, the R factor is defined [97], given experimentally determined structure factors F_{obs} and model-determined structure factors F_{calc}, as:

$$R = \frac{\sum_{i,j,k} \left| \left| F_{obs}^{(i,j,k)} \right| - \left| F_{calc}^{(i,j,k)} \right| \right|}{\sum_{i,j,k} \left| F_{obs}^{(i,j,k)} \right|} \tag{2.1}$$

The R factor measures how well the proposed 3D structure *explains* the observed electron-density data. Crystallographers usually strive to get R factors under 0.2 (or lower, depending on map resolution), while also building a physically feasible (i.e. valid bond distances, torsion angles, etc.) protein model, all without adding too many free water molecules. One can always reduce

the R factor by placing extra water molecules in the density map; these reductions are a result *overfitting* the model to the data, and do not correspond to a physically feasible interpretation.

Another commonly used measure is *free R factor*, or R_{free} [10]. Here, 5-10% of reflections are randomly held out as a test set before refinement. R_{free} is the R factor for these held-aside reflections. Using R_{free} tends to avoid overfitting the protein's atoms to the reflection data.

2.1.3 Algorithmic background

Algorithms for automatically interpreting electron-density maps draw heavily from the machine learning and statistics communities. These communities have developed powerful frameworks for modeling uncertainty, reasoning from prior examples, and statistically modeling data, all of which have been used by researchers in crystallography. This section briefly describes the statistical and machine-learning methods used by the interpretation methods presented throughout my thesis. This section is intended as a basic introduction to these topics. Russell and Norvig's text [99] or Mitchell's text [82] provides a thorough overview of these topics.

2.1.3.1 Probabilistic models

A *model* here refers to a system that simulates a real-world event or process. Probabilistic models simulate uncertainty by producing different outcomes with different probabilities. In such models, the probabilities associated with certain events is generally not known, and instead has to be estimated from a *training set*, a set of previously solved problem instances. Using *maximum likelihood estimation*, the probability of a particular outcome is estimated as the frequency at which that outcome occurs in the training set.

The *unconditional* or *prior probability* of some outcome A is denoted $P(A)$. Assigning some value to this probability, say $P(A) = 0.3$, means that in the absence of any other information, the best assignment of probability of outcome A is 0.3. As an example, when flipping a (fair) coin, $P(\text{“}heads\text{”}) = 0.5$. In this section, I use "outcome" to mean the setting of some random variable; $P(X = x_i)$ is the probability that variable X takes value x_i. Throughout my thesis, I will use the shorthand $P(x_i)$ to refer to this value.

The *conditional* or *posterior probability* is used when other, previously unknown, information becomes available. If other information, B, relevant to event A is known, then the best assignment of probability to event A is given by the conditional $P(A|B)$. This reads as "the probability of A, *given* B." If more evidence, C, is uncovered, then the best probability assignment is $P(A|B, C)$ (where "," denotes "and").

The *joint probability* of two or more events is the probability of both events occurring, and – for two events A and B – is denoted $P(A, B)$ and is read as "the probability of A *and* B". Conditional and joint probabilities are related using the expression:

$$P(A, B) = P(A|B)P(B) = P(B|A)P(A) \tag{2.2}$$

This relation holds for any events A and B. Two events are *independent* if their joint probability is the same as the product of their unconditional probabilities, $P(A, B) = P(A)P(B)$. If A and B are independent one also has $P(A|B) = P(A)$, that is, knowing B tells us nothing about A.

One computes the *marginal probability* by taking the joint probability and summing out one or more variables. That is, given the joint distribution $P(A, B, C)$, one computes the marginal distribution of A as:

$$P(A) = \sum_B \sum_C P(A, B, C) \tag{2.3}$$

Above, the sums are over all possible outcomes of events B and C. The marginal distribution is important because it provides information about the distribution of some variables (A above) in the full joint distribution, without requiring one to explicitly compute the (possibly intractable) full joint distribution.

Finally, *Bayes' rule* allows one to reverse the direction of a conditional:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{2.4}$$

Bayes' rule is useful for computing a conditional $P(A|B)$ when direct estimation (using frequencies from a training set) is difficult, but when $P(B|A)$ can be estimated accurately. Often, one drops the denominator, and instead computes the *relative likelihood* of two outcomes, for example, $P(A = a_1|B)$ versus $P(A = a_2|B)$. If a_1 and a_2 are the only possible outcomes for A, then exact probabilities can be determined by normalization; there is no need to compute $P(B)$.

2.1.3.2 Case-based reasoning

Broadly defined, *case-based reasoning* (CBR) attempts to solve a new problem by using solutions to similar past problems. Algorithms for case-based reasoning require a database of previously solved problem instances, and some distance function to calculate how "different" two problem instances are. There are two key aspects of CBR systems. First, learning in such systems is *lazy*: the models only generalize to unseen instances when presented with such a new instance. Second, they only use instances "close" to the unseen instance when categorizing it.

The most common CBR algorithm is *k-nearest neighbor* (kNN). In kNN, problem instances are typically feature vectors, that is, points in some n-dimensional space. The learning algorithm, when queried with a new problem instance $S = \langle s_1, \ldots, s_n \rangle$ for classification or regression, finds the k previously solved problem instances closest to the query in Euclidean space. That is, one chooses the examples minimizing the distance:

$$d(S, T) = \sqrt{\sum_{i=1}^{n} (s_i - t_i)^2} \tag{2.5}$$

Then, the k neighbors "vote" on the query instance's label: usually the majority class label (for classification) or average label (for regression) of the k neighbors is used. One variant of kNN weights each neighbor's vote by its similarity to the query. Another variant learns weights for each dimension, to be used when computing the distance between two instances.

2.1.4 Undirected graphical models

Although the topic of undirected graphic models could be placed in the "computational background" subsection, it is central enough to my thesis to warrant its own subsection. Graphical models, such as Bayesian networks and Markov fields, represent the joint probability distribution over a set of variables as a function defined over some graph. A *pairwise undirected graphical model* (or *pairwise Markov field*) constructs an undirected graph where each vertex is associated with one or more hidden variables. Given this model, the probability of some setting of the random variables is the product of *potential functions* defined on each *edge* and *vertex* in the graph.

Formally, the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of nodes $s \in \mathcal{V}$ connected by edges $(s,t) \in \mathcal{E}$. Each node in the graph is associated with a (hidden) random variable $x_s \in \mathbf{x}$, and the graph is conditioned on a set of observation variables $\mathbf{y}$. For object recognition, these x_s's are the 3D position of part s. Each vertex has a corresponding *observation potential* $\psi_s(x_s, \mathbf{y})$, and each edge is associated with a *structural potential* $\psi_{st}(x_s, x_t)$. Then, one represents the full joint probability as:

$$p(\mathbf{x}|\mathbf{y}) \propto \prod_{(s,t)\in\mathcal{E}} \psi_{st}(x_s, x_t) \times \prod_{s\in\mathcal{V}} \psi_s(x_s|\mathbf{y}) \tag{2.6}$$

In many applications, one is most concerned with finding the *maximum marginal assignment*, that is, the labels $x_s \in \mathbf{x}$ that maximize this joint probability for some value of $\mathbf{y}$.

Figure 2.7 illustrates a simple Markov field model with just 5 vertices. Given that each vertex i is associated with a single variable x_i, then the probability of some setting of these five variables is:

$$P(x_1, x_2, x_3, x_4, x_5) = \frac{1}{Z} \times \psi_1(x_1) \times \psi_{12}(x_1, x_2) \times \psi_2(x_2) \times \psi_{13}(x_1, x_3) \times \psi_2(x_3) \times \\ \times \psi_{24}(x_2, x_4) \times \psi_{34}(x_3, x_4) \times \psi_4 x_4 \times \psi_{45}(x_4, x_5) \times \psi_5(x_5)$$

The function Z (arguments omitted for clarity) is the *partition function*, and is used to normalize the product of potential functions to sum to unity over all possible settings of every variable.

2.1.4.1 Inference in undirected graphical models

Given a graphical model, *inference* attempts to find the most-probable setting of each of the variables in the model. Several different inference algorithms have been successfully used in undirected graphical models. If the graph (as well as the state space of each variable) is small, then exhaustive enumeration of states may be sufficient. If the graph is large but has no loops, then dynamic programming will infer both marginal probabilities, using the forward algorithm, and the maximum-likelihood path, using the Viterbi algorithm [114]. Finally, if the graph is large and has loops, then exact methods will not work, and one is forced to use approximate inference algorithms. Three types of algorithms that are commonly used include: (a) Monte Carlo algorithms, such as Markov-chain Monte Carlo [88] and Gibbs sampling [36], (b) variational methods [61], and (c) (loopy) belief propagation [78, 93].

A variety of Monte Carlo algorithms have been developed and applied to probabilistic inference problems [72,91]. These algorithms use sampling from a Markov chain to explore high-probability

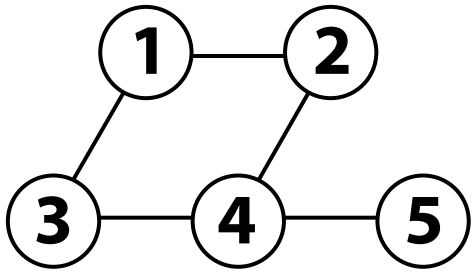


Figure 2.7 A simple pairwise Markov-field model.

regions in state space. These algorithms, which include the Metropolis-Hastings algorithm [42], Gibbs sampling [4, 36], and simulated annealing [65], are attractive because of their simplicity of implementation and guaranteed convergence, but are often slow to converge.

Variational methods formulate inference as an optimization problem; solving the optimization problems gives bounds on some probability of interest. Variational methods refer not to a single approach, but rather a whole class of methods, including mean-field methods [95], variational EM [89], variational Bayes [120], and variational PCA [5]. Further discussion of these methods are well beyond the scope of this thesis; interested readers will want to consult Jordan's tutorial on variational methods [61] or Bishop's text [6].

Finally, belief propagation (BP) is a message-passing algorithm, originally proposed by Pearl [93] for exact inference in tree-structured model. Loopy belief propagation is the application of Pearl's polytree algorithm to graphs with arbitrary topologies. There are no guarantees of convergence – and convergence may not be to anything reasonable – but it has been shown to work well in practice [85].

2.1.4.2 Belief propagation

Belief propagation is central to the inference methods presented my thesis; this section presents BP in more detail. In addition to this section, Section 3.3.2 illustrates BP graphically (using the framework introduced in Chapter 3).

Belief propagation – based on Pearl's polytree algorithm [93] – computes the *marginal probability* over each random variable x_s by passing a series of local messages. Recall that the marginal probability refers to the joint probability, where all but one variable is summed out, that is:

$$b_s(x_s|\mathbf{y}) = \sum_{x_1} \cdots \sum_{x_{s-1}} \sum_{x_{s+1}} \cdots \sum_{x_N} P(\mathbf{x}|\mathbf{y}) \tag{2.7}$$

Pseudocode for BP inference appears in Algorithm 2.1. At each iteration, a vertex in the graph computes the product of all incoming messages, then passes a *convolution* of this product to its

Algorithm 2.1 Belief propagation.

input: Observational potentials $\psi_s(x_s|\mathbf{y})$ and structural potentials $\psi_{st}(x_s, x_y)$

output: An approximation to the marginal $\hat{b}_s(x_s|\mathbf{y}) \approx \sum_{x_1} \dots \sum_{x_{s-1}} \sum_{x_{s+1}} \dots \sum_{x_N} P(\mathbf{x}|\mathbf{y})$

```
// Initialize messages to uniform
```
initialize messages to 1

```
// Repeat until convergence (or until some iteration limit)
```
while $\hat{b}$*'s have not converged* **do**
```
   // For each vertex in the graph (in some predetermined order)
```
   foreach *part* $s = 1 \dots N$ **do**
```
      // Set s's belief to its vertex potential
```
      $\hat{b}_s^n(x_s|\mathbf{y}) \leftarrow \psi_s(x_s|\mathbf{y})$
```
      // For each neighbor of s in the undirected graph
```
      foreach *vertex* $t \in \Gamma(s)$ **do**
```
         // If neighbor t's belief has been updated since the last
         //     message computation, update the message from t to s
```
         if $\hat{b}_t^n$ *has been updated* **then**
            $m_{t\to s}^n(x_s) \leftarrow \int_{x_t} \psi_{st} \times (\hat{b}_t^n / m_{s\to t}^{n-1})\, dx_t$
         end
```
         // Multiply s's belief by the incoming message from t
```
         $\hat{b}_s^n(x_s|\mathbf{y}) \leftarrow \hat{b}_s^n(x_s|\mathbf{y}) \times m_{t\to s}^n(x_s)$
      end
   end
end

neighbors (for clarity, the message's dependence on $\mathbf{y}$ is usually dropped):

$$m_{t\to s}^n(x_s) \propto \int_{x_t} \psi_{st}(x_s, x_t) \times \psi_t(x_t|\mathbf{y}) \times \prod_{u\in\Gamma(t)\setminus s} m_{u\to t}^{n-1}(x_t)\, dx_t \tag{2.8}$$

Here, $\Gamma(t)\setminus s$ denotes all neighbors of t in the graph *excluding* s. Messages are normalized so that the probabilities sum to unity. Koller *et al.* [66] suggest assigning some order to the nodes, and updating the belief at each node sequentially (though recent evidence suggests dynamic ordering may be preferable [108]). At any iteration, the algorithm computes an approximation to the marginal as the product of incoming messages and the node's observation potential ψ_s,

$$\hat{b}_s^n(x_s|\mathbf{y}) \propto \psi_s(x_s|\mathbf{y}) \times \prod_{u\in\Gamma(t)} m_{u\to t}^n(x_t) \tag{2.9}$$

In tree-structured graphs (graphs without cycles), this algorithm is exact. Unfortunately, for many tasks, this limitation is overly restrictive. This algorithm may be applied to graphs with

arbitrary topologies; however, there are no guarantees to the convergence of this algorithm – and convergence may not be to the correct solution – but empirical results show that "loopy BP" often produces good estimates in practice [85, 118].

Several papers have explored circumstances under which loopy BP's convergence or optimality can be guaranteed. Weiss has shown a category of graphical models with a single loops in which optimality is guaranteed [117]. More recent work [121] has shown the existence of fixed-points in loopy BP, but they are neither unique nor optimal. Heskes [47] has developed sufficient conditions for the uniqueness of BP's fixed-points. Others have characterized the fixed-points in loopy BP [109].

Others have explored message approximation in loopy BP. When exact message computation is intractable, stochastic approximation of messages [66] as well as message simplification [14] have been investigated. Additionally, when dealing with continuous-valued variables, some sort of approximation or simplifying assumptions must be made [56, 105]. Ihler *et al.* [52] have explored the consequences of approximating messages in BP, placing bounds on accumulated message errors as BP progresses.

A recent paper [105] investigates the special case where the labels x_t are continuously valued. Using ideas from particle filtering [29], these authors develop nonparametric belief propagation (NBP), which uses weighted Gaussian probability density estimates for both messages and beliefs. That is, given a set of weights w_s^i, $i = 1 \ldots N$, a set of Gaussian centers μ_s^i and a covariance matrix Λ_s, an estimate of the belief is given by

$$\hat{b}_s^n(x_s|\mathbf{y}) = \sum w_s^{(i)} \times \mathcal{N}(x_s; \mu_s^{(i)}, \Lambda_s) \tag{2.10}$$

Their message computation uses an efficient Gibbs sampling routine. The Gibbs sampler [52] approximates the product of k Gaussian mixtures – each with M components – as an M-component mixture. This sampling is used to compute BP message products. For the BP convolution operation, forward sampling is employed. Their inference algorithm was applied to several vision tasks. Isard [56] makes use of a similar technique, with a sampling routine specialized to mixture-of-Gaussian edge potentials.

2.2 Alternative approaches to automatic density-map interpretation

Several research groups have investigated automating the interpretation of electron-density maps. This section presents a high-level overview of some of these methods, while the remainder of this chapter takes an in-depth look at three of these methods, describing algorithmically how they have approached this problem.

By far the most commonly used method is ARP/wARP [71,83,94]. This "atom-based" method heuristically places atoms in the map, connects them, and refines their positions. To handle poor phasing, ARP/wARP uses an iterative algorithm, consisting of alternating phases in which (a) a model is built from a density map, and (b) the density map's phases are improved using the constructed model. This algorithm is widely used, but has one drawback: fairly high resolution data,

about 2.3Å or better, is needed. Given this high-resolution data, the method is extremely accurate; however, many protein crystals fail to diffract to this extent.

Several approaches represent the density map in an alternate form, in the process making the map more easily interpretable for both manual and automated approaches. One of the earliest such methods, *skeletonization*, was proposed for use in protein crystallography by Greer's BONES algorithm [40]. Skeletonization, similar to the medial-axis transformation in computer vision, gradually thins the density map until it is a narrow ribbon approximately tracing the protein's backbone and sidechains. More recent work by Leherte *et al.* [73], represents the density map as an acyclic graph: a minimum spanning tree connecting all the critical points (points where the gradient of the density is 0) in the electron-density map. This representation accurately approximates the backbone when given 3Å or better data, and separates protein from solvent up to 5Å resolution.

Cowtan's FFFEAR efficiently locates rigid templates in the density map [17]. It uses *fast Fourier transforms* (FFTs) to quickly match a learned template over all locations in a density map. Cowtan provides evidence showing it locates alpha helices in poorly-phased 8Å maps. Unfortunately, the technique is limited in that in can only locate large rigid templates (e.g. those corresponding to secondary-structure elements). One must trace loops and map the fit to the sequence manually. However, several methods use FFFEAR as a template-matching subroutine in a larger interpretation algorithm.

X-AUTOFIT, part of the QUANTA [90] package, uses a gradient-refinement algorithm to place and refine the protein's backbone. Their refinement takes into account the density map as well as bond geometry constraints. They report successful application of the method at resolutions ranging from 0.8 to 3.0Å.

Terwilliger's RESOLVE contains an automated model-building routine [110, 111]. It uses a hierarchical procedure in which helices and strands are located and fitted, then are extended in an iterative fashion using a library of tripeptides. Finally, RESOLVE applies a greedy fragment-merging routine to overlapping extended fragments. The approach is able to place approximately 50% of the protein chain in a 3.5Å resolution density map.

Levitt's MAID [75] approaches map interpretation "as a human would," by first finding the major secondary structures, alpha helices and beta sheets, connecting the secondary-structure elements, and mapping this fit to the provided sequence. MAID reports success on density maps at around 2.8Å resolution.

Ioerger's TEXTAL [38, 53, 54] attempts to interpret poor-resolution (2.2 to 3.0Å) density maps using ideas from pattern recognition. Ioerger constructs a set of 15 rotation-invariant density features. Using these features at several radii, a subroutine, CAPRA, trains a neural network to identify Cα atoms. TEXTAL then identifies sidechains by looking at the electron-density around each putative alpha carbon, efficiently finding the most similar region in a database, and placing into the model the corresponding sidechain.

Finally, ACMI, the algorithm introduced in this thesis, takes a probabilistic approach to density map interpretation [27]. Residues of the protein are modeled as nodes in a graph, while edges model pairwise structural interactions arising from chemistry. An efficient inference algorithm determines the most probable backbone trace conditioned on these interactions. A particle-filtering

algorithm places individual sidechains, growing the model one amino acid at a time. ACMI locates accurate models in 3.0 to 4.0Å density maps.

The rest of this chapter further describes three of these methods – ARP/wARP, RESOLVE and TEXTAL – in detail. Each section presents a method, describing strengths and weaknesses. High-level pseudocode clarifies important subroutines. Throughout the chapter, a small running example is employed to illustrate intermediate steps of the various algorithms. The example uses the density map of protein 1XMT, a 95-amino-acid protein with two symmetric copies in the unit cell. The 2.5Å density map and its crystallographer-determined solution appears in Figure 2.8 (masked so that only one symmetric copy is displayed).

The running example is not meant as a test of the algorithms, but rather as a real-world illustrative example. The example map is natively at 1.15Å resolution. Full native resolution is used for ARP/wARP; the map is artificially downsampled to 2.5Å resolution for RESOLVE and for TEXTAL (as these algorithms were designed for poor-resolution data).

2.2.1 ARP/wARP

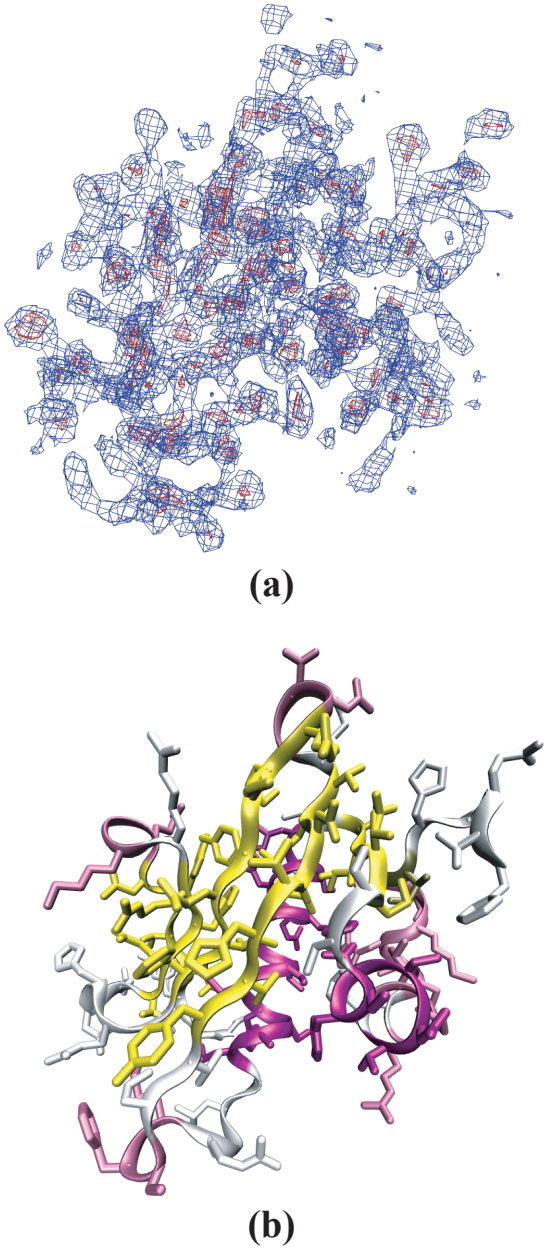
The ARP/wARP (automated refinement procedure) software suite is a crystallographic tool for the interpretation and refinement of electron-density maps. ARP/wARP's WARPNTRACE procedure was the first automatic interpretation tool successfully used for protein models. Today, it remains one of the most used tools in the crystallographic community for 3D protein-image interpretation. ARP/wARP concentrates on the best placement of individual atoms in the map: no attempt is made to identify higher-order constructs like residues, helices, or strands. ARP/wARP's "atom-level" method requires high-quality data, however. In general, ARP/wARP requires maps at a resolution of 2.3Å or higher to produce an accurate trace.

Figure 2.9 illustrates an overview of WARPNTRACE. WARPNTRACE begins by creating a *free atom model* – a model containing only unconnected atoms – to fill in the density map of the protein. It then connects some of these atoms using a heuristic, creating a *hybrid model*. This hybrid model consists of a partially-connected backbone, together with a set of unconstrained atoms. This hybrid model is refined, producing a map with improved phase estimates. The process iterates using this improved map. At each iteration, WARPNTRACE removes every connection, restarting with a free-atom model.

2.2.1.1 Free-atom placement

ARP/wARP contains an atom-placement method based on ARP, an interpretation method for general molecular models. ARP randomly places unconnected atoms into the density map, producing a *free-atom model*, illustrated in Figure 2.10a.

To initialize the model, ARP begins with a small set of atoms in the density map. It slowly expands this model by looking for areas above a density threshold, at a bonding distance away from existing atoms. The density threshold is slowly lowered until ARP places the desired number of free atoms.



(a)

(b)

Figure 2.8 (a) The 2.5Å resolution electron-density map of the 95-amino-acid protein 1XMT. (b) The crystallographer-determined solution. For clarity, the backbone is modeled using a thicker ribbon, while the thinner segments indicated bonds between sidechain atoms. The model is colored by structure (α-helices/β-sheets shaded and loops white). This density map, at several resolutions, will be used as a running example throughout the section.

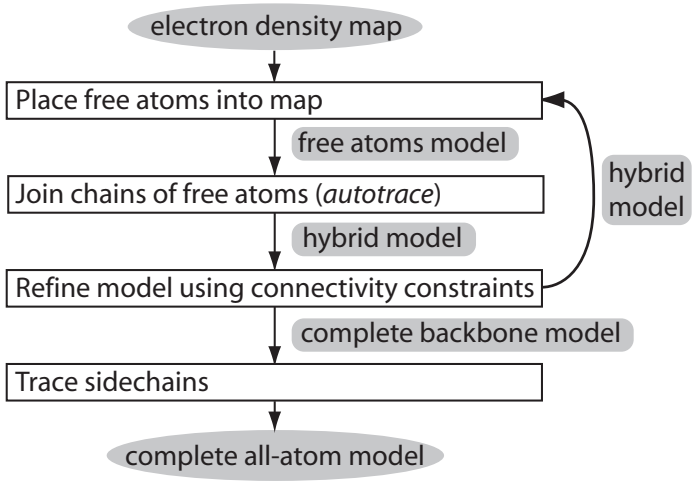


Figure 2.9 A flowchart of ARP/wARP's WARPNTRACE.

For small molecules, ARP's next step is *refining* the free-atom model; that is, iteratively moving atoms to better explain the density map. Free-atom refinement ignores stereochemical information, and moves each atom independently to produce a complete structure. ARP's free-atom refinement, in addition to moving atoms, considers adding or removing atoms from the model. Multiple randomized initial structures are used to improve robustness. Further details are available from Perrakis *et al.* [94].

However, with molecules as large as proteins, free-atom refinement alone is insufficient to produce an accurate model. Performing free-atom refinement with tens of thousands of atoms leads to overfitting, producing a model that is not physically feasible. For determining protein structures, ARP/wARP makes use of connectivity information in its refinement, using free-atom placement as a starting point. The procedure WARPNTRACE adds connectivity information to the free atom model.

2.2.1.2 Backbone tracing

Given a free-atom model of a protein, one can form a crude backbone trace by looking for pairs of free atoms the proper distance apart. WARPNTRACE formalizes this procedure, called *autotracing*, using a heuristic method. The method is outlined in Algorithm 2.2. WARPNTRACE assigns a score – based on density values – to each free atom. The highest scoring atom pairs 3.8 ± 0.5Å apart become candidate Cα's. The algorithm verifies candidate pairs by overlaying them with a peptide template. If the template matches the map, WARPNTRACE saves the candidate pair.

After computing a list of Cα pairs, WARPNTRACE constructs the backbone using a database of known backbone conformations (taken from solved protein structures). Given a chain of candidate

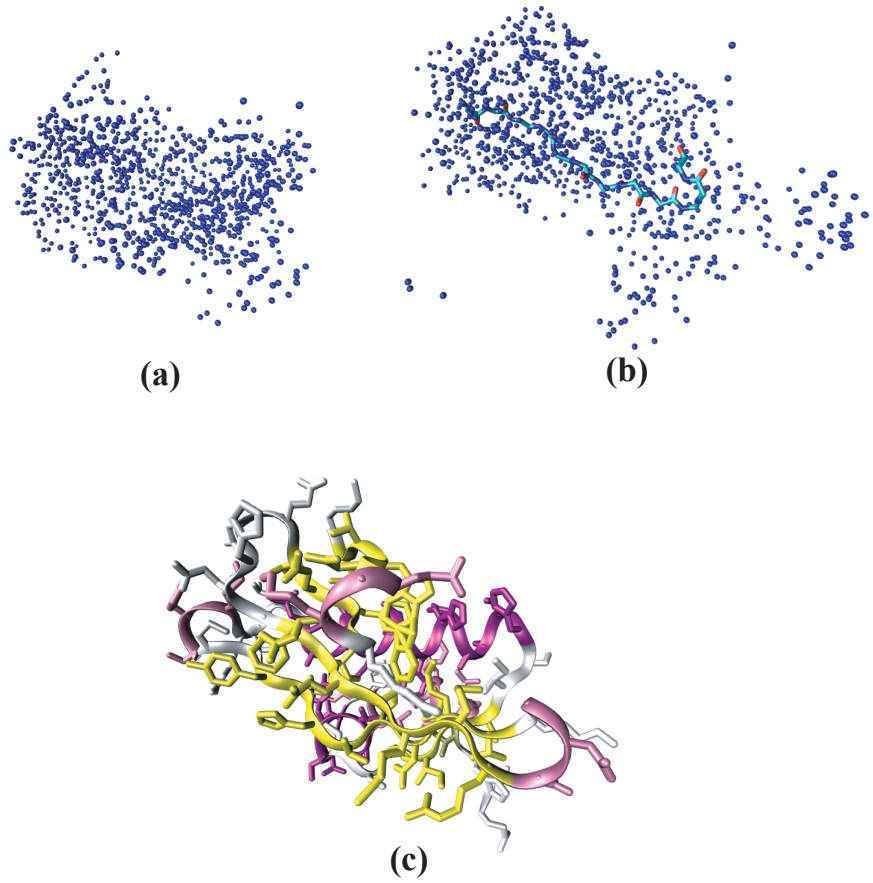


Figure 2.10 Intermediate steps in ARP/wARP's structure determination: (a) the free atom model, where waters are placed in the map, (b) the hybrid model, after some connections are determined, and (c) the final predicted structure.

Cα pairs, WARPNTRACE considers all backbone conformations in the database with matching Cα positions, ordered by length. The longest candidate backbone is then added to the model. The algorithm connects the corresponding free atoms, and removes these atoms from the free-atom pool. The process repeats as long as there remain candidate Cα chains at least 5 residues in length.

Autotracing produces a *hybrid model*, shown in Figure 2.10b. A hybrid model contains a set of connected chains together with a set of free atoms. Autotracing identifies some atom types and connectivity, which enables the use of some stereochemical information in refinement. Added restraints increase the number of observations available, and increase the probability of producing a good model. The tracing is initially very conservative, with many free atoms remaining in the model. Adding too many restraints too early leads to overfitting the model.

Finally, a modified version of ARP refines this hybrid model. ARP uses the refined structure to improve the experimentally determined phases, making the map clearer to interpret. At

Algorithm 2.2 WARPNTRACE's model-building algorithm.

input: electron-density map $\mathbf{M}$, free-atom model $\mathbf{F}$, sequence $\mathbf{seq}$
output: all-atom protein model

for $i = 1$ *to* $nIterations$ **do**
  `// initialize hybrid model`
  $\mathbf{H} \leftarrow \mathbf{F}$
  `// find candidate C`α `pairs`
  $\mathbf{CA\_pairs} \leftarrow$ highest-scoring atom pairs 3.8 ± 0.5Å apart
  `// verify C`α `pairs by matching to a template`
  foreach $c_i \in \mathbf{CA\_pairs}$ **do**
    if *(c_i does not match backbone template)* **then**
      delete c_i from $\mathbf{CA\_pairs}$
    end
  end
  `// connect long chains in hybrid model`
  while *a C*α *chain of length* ≥ 5 *remains in* $\mathbf{CA\_pairs}$ **do**
    $bestChain \leftarrow$ longest fragment in DB overlapping $\mathbf{CA\_pairs}$
    remove $bestChain$'s atoms from $\mathbf{CA\_pairs}$, $\mathbf{H}$
    add $bestChain$ to $\mathbf{H}$
  end
  `// refine hybrid model`
  $\mathbf{H}' \leftarrow refine(\mathbf{H})$
  $\mathbf{F} \leftarrow$ remove connections from hybrid model $\mathbf{H}'$
end
$model \leftarrow$ sidechainTrace($\mathbf{H}'$)

each iteration of this "autotrace–refine–recompute phases" cycle, WARPNTRACE returns to a free-atom model, by removing previous connections. Since the map is better-phased, autotracing may produce a more complete model. This, in turn, provides a better refinement, further improving phasing.

This cycle continues for a fixed number of iterations, or until a complete trace is available. Finally, WARPNTRACE adds on side-chains by identifying patterns of free atoms around each Cα. It aligns these free-atom patterns to the sequence to produce a complete model. Figure 2.10c illustrates the complete ARP/wARP-determined trace on our running example.

2.2.1.3 Discussion

ARP/wARP is the preferred method for automatically interpreting electron-density maps, assuming sufficiently high-resolution data is available. Its placement of individual atoms, followed by atom-level refinement, produces an extremely accurate trace with no user action required in

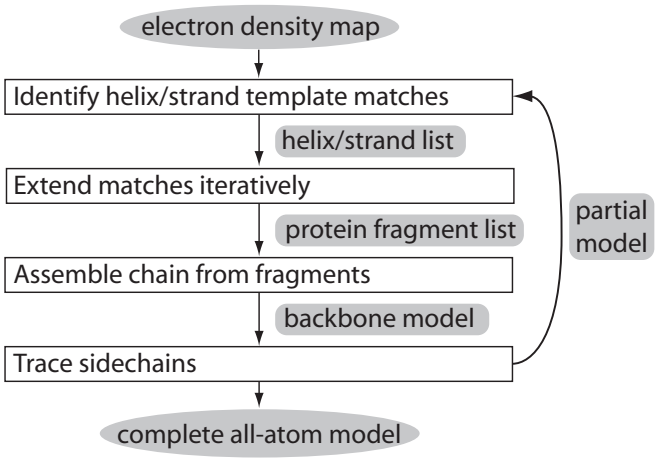


Figure 2.11 A flowchart of RESOLVE.

2.3Å or better density maps. It is widely used by crystallographers to rapidly construct a protein model. Unfortunately, many protein crystals fail to produce maps of sufficient quality, and one must consider alternate methods.

2.2.2 RESOLVE

While ARP/wARP is extremely accurate with high-resolution data, many protein crystals fail to diffract to a sufficient level for accurate interpretation. In general, ARP/wARP requires individual atoms to be visible in the density map, which happens at about 2.3Å resolution or better. The next two methods – RESOLVE and TEXTAL – both aim to solve maps with > 2.3Å resolution. Both methods take a different approach to the problem; however, both – in contrast to ARP/wARP – consider higher-level constructs than atoms when building a protein model. This allows accurate interpretation even when individual atoms are not visible.

RESOLVE is a method developed by Terwilliger for automated model-building in poor-quality electron-density maps [110, 111]. Figure 2.11 outlines the complete hierarchical approach. RESOLVE's method uses two (learned) model secondary-structure fragments – a short α-helix and β-strand – for its initial matching, searching over all rotations and translations of these fragments. After placing a small set of overlapping model fragments into the map, the algorithm considers a much larger template set as potential extensions. RESOLVE joins overlapping fragments and, finally, identifies sidechains corresponding to each Cα, conditioned on the input sequence, and places individual atoms into the model.

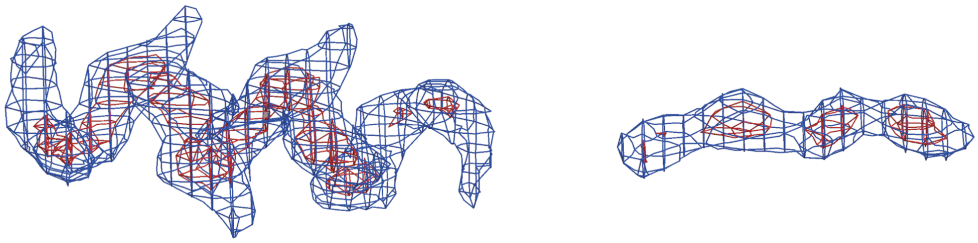


Figure 2.12 The averaged helix (left) and strand (right) fragment used in RESOLVE's initial matching step.

2.2.2.1 Secondary-structure search

Given an electron-density map, RESOLVE begins its interpretation by searching all translations and rotations in the map for a model 6-residue α-helix and a model 4-residue β-strand. RESOLVE constructs these fragments by aligning a collection of helices (or strands) from solved structures; it computes the electron-density for each at 3Å resolution, and averages the density across all examples. The "average" models used by RESOLVE are illustrated in Figure 2.12.

Given these model fragments, RESOLVE considers placing them at each position in the map. At each position it considers all possible rotations (at a 30° or 40° discretization) of the fragment, and computes a standardized squared-density difference between the fragment's electron-density and the map:

$$t(\vec{x}) = \sum_{\vec{y}} \epsilon_f(\vec{y}) \left(\rho'_f(\vec{y}) - \frac{1}{\sigma_f(\vec{x})} \left[\rho(\vec{y} - \vec{x}) - \bar{\rho}(\vec{x}) \right] \right) \tag{2.11}$$

Above, $\rho(\vec{x})$ is the map in which I am searching, $\rho'_f(\vec{x})$ is the *standardized* fragment electron-density, $\epsilon_f(\vec{x})$ is a masking function that is nonzero only for points near the fragment, and $\bar{\rho}(\vec{x})$ and $\sigma_f(\vec{x})$ standardize the map in the masked region ϵ_f centered at $\vec{x}$:

$$\begin{aligned} \bar{\rho}(\vec{x}) &= \frac{\sum_{\vec{y}} \epsilon_f(\vec{y}) \rho(\vec{y} - \vec{x})}{\sum_{\vec{y}} \epsilon_f(\vec{y})} \\ \sigma_f^2(\vec{x}) &= \frac{\sum_{\vec{y}} \epsilon_f(\vec{y}) \left[\rho(\vec{y} - \vec{x}) - \bar{\rho}(\vec{x}) \right]^2}{\sum_{\vec{y}} \epsilon_f(\vec{y})} \end{aligned} \tag{2.12}$$

RESOLVE computes the matching function $t(\vec{x})$ quickly over the entire unit cell by FFT convolution using FFFEAR [17].

After matching the two model fragments using a coarse rotational step-size, the method generates a list of best-matching translations and orientations of each fragment (shown in Figure 2.13a).

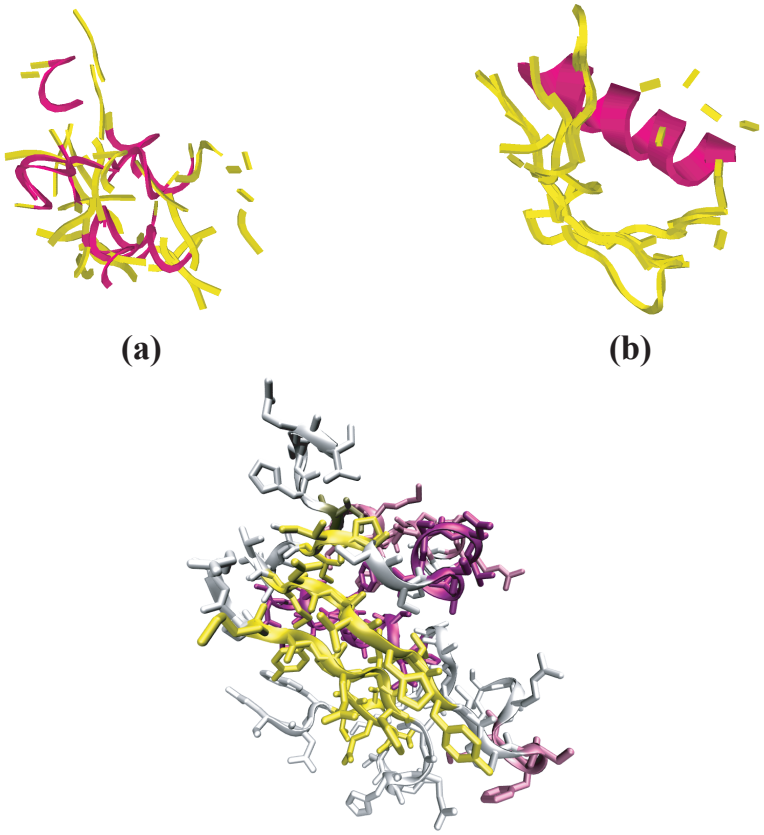


Figure 2.13 Intermediate steps in RESOLVE's structure determination: (a) locations in the map that match short helical/strand fragments, (b) the same fragments after refinement and extension, and (c) the final predicted structure.

Processing these matches in order, RESOLVE refines each fragment's position and rotation to maximize the real-space correlation coefficient (RSCC) between template and map:

$$RSCC(\rho_f, \rho) = \frac{\langle \rho_f \cdot \rho \rangle - \langle \rho_f \rangle \langle \rho \rangle}{\sqrt{\langle \rho_f^2 \rangle - \langle \rho_f \rangle^2} \sqrt{\langle \rho^2 \rangle - \langle \rho \rangle^2}} \tag{2.13}$$

Here, $\langle \rho \rangle$ indicates the map mean over a fragment mask. RESOLVE only considers refined matches with an RSCC above some threshold.

2.2.2.2 Iterative fragment extension

At this point, RESOLVE has a set of putative helix and strand locations in the density map. The next phase of the algorithm extends these using a much larger library of fragments, producing a model like that in Figure 2.13b. Specifically, RESOLVE makes use of four such libraries for fragment extension:

(a) 17 α-helices between 6 and 24 amino acids in length

(b) 17 β-strands between 4 and 9 amino acids in length

(c) 9,232 tripeptides containing backbone atoms only *for N-terminus extension*

(d) 4,869 tripeptides containing a partial backbone (the chain $C_\alpha - C - O$ with no terminal N) plus two full residues *for C-terminus extension*

RESOLVE learns these fragment libraries from a set of previously solved protein structures. It constructs the two tripeptide libraries by clustering a larger dataset of tripeptides.

α-helix/β-strand extension. For each potential model's placement in the map, RESOLVE considers extending it using each fragment in either set (a), if the model fragment is a helix, or set (b), if the model fragment is a strand. For each fragment, RESOLVE chooses the longest segment of the fragment such that every atom in the fragment has a density value above some threshold.

To facilitate comparison between these 17 segments of varying length (one for each fragment in the library), each matching segment is given a score $Q = \langle\rho\rangle\sqrt{N}$, with $\langle\rho\rangle$ the mean atom density, and N the number of atoms. The algorithm computes a Z-score:

$$Z = \frac{Q - \langle Q\rangle}{\sigma(Q)} \tag{2.14}$$

RESOLVE only considers segments with $Z > 0.5$. Notice there may be a large number of overlapping segments in the model at this point.

Loop extension using tripeptide libraries. For each segment in the model, RESOLVE attempts to extend the segment in both the N-terminal and C-terminal direction using the tripeptide library. RESOLVE tests each tripeptide in the library by superimposing the first residue of the tripeptide on the last residue of the current model segment. It then tests the top scoring "first-level" fragments for overlap with the current model segment (i.e., self-collisions). For those with no overlap, a look-ahead step considers this first-level extension as a starting point for a second extension. The score for each first-level extension is:

$$score_{\text{first-level}} = \langle\rho_{\text{first-level}}\rangle + \max_{\text{second-level}} \langle\rho_{\text{second-level}}\rangle \tag{2.15}$$

Above, $\langle\rho_{\text{first-level}}\rangle$ denotes the average map density at the atoms of the first-level extension.

RESOLVE accepts the best first-level extension – taking the look-ahead term into account – only if the average density is above some threshold density value. If the density is too low, and

Algorithm 2.3 RESOLVE's chain-assembly algorithm.

input: electron-density map $\mathbf{M}$, set of high scoring fragments $\mathbf{F}$
output: putative backbone trace $\mathbf{X} = \{\vec{x}_i\}$ including C_β positions

```
// extend matches until no candidate chains remain
```
repeat
  $frag_{best} \leftarrow$ top scoring unused segment
  `// look for pairs of chains with overlapping Cα's`
  for *each* $frag_i \in \{\mathbf{F} \backslash frag_{best}\}$ **do**
    if $frag_i$ *and* $frag_{best}$ *overlap at* ≥ 2 *C*α *positions*
    and *extension does not cause steric clashes* **then**
      extend $frag_{best}$ by $frag_i$
    end
  end
until *no candidates remain*

the algorithm rejects the best fragment, several "backup procedures" consider additional first-level fragments, or stepping back one step in the model segment. If these backup procedures fail, RESOLVE rejects further extensions.

2.2.2.3 Chain assembly

Given this set of candidate model segments, RESOLVE next assembles a continuous chain. To do so, it uses an iterative method, outlined in Algorithm 2.3. The outermost loop repeats until no more candidate segments remain. At each iteration, the algorithm chooses the top-scoring candidate segment not overlapping any others. It considers all other segments in the model as extensions: if at least two Cα's between the candidate and extension overlap, then RESOLVE accepts the extension. Once accepted, the extension becomes the current candidate.

2.2.2.4 Sidechain placement

RESOLVE's final step is, given a set of Cα positions in some density map, to identify the corresponding residue type, and to trace all the sidechain atoms [111]. This sidechain tracing is the first time that RESOLVE uses the protein's sequence. RESOLVE's sidechain tracing uses a probabilistic method, finding the most likely layout conditioned on the input sequence. RESOLVE's sidechain tracing procedure is outlined in Algorithm 2.4.

RESOLVE's sidechain tracing relies on a rotamer library. This library consists of a set of low-energy conformations – or *rotamers* – that characterizes each amino-acid type. RESOLVE builds a rotamer library from the sidechains in 574 protein structures. Clustering produces 503 different low-energy side-chain conformations. For each cluster member, the algorithm computes a density map; each cluster's representative map is the average of its members.

For each Cα, RESOLVE computes a probability distribution of the corresponding residue type. Probability computation begins by first finding the correlation coefficient (see Equation 2.13) between the map and each rotamer. For each rotamer j, the correlation coefficient at the kth Cα is given by cc_{jk}. A Z-score is computed, based on rotamer j's correlation at every other Cα:

$$Z_{jk}^{rot} = \frac{cc_{jk} - \langle cc_j \rangle}{\sigma_j} \tag{2.16}$$

The algorithm only keeps a *single best-matching rotamer* of each residue type. That is, for residue type i:

$$Z_{ik}^{res} = \max_{\text{fragment } j \text{ is of type } i} Z_{jk}^{rot} \tag{2.17}$$

RESOLVE uses a Bayes rule to compute a probability from the Z-score. Amino-acid distributions in the input sequence provide the *a priori* probability P_{0j} of residue type j. Given a set of correlation coefficients at some position, RESOLVE computes the probability that the residue type is i by taking the product of probabilities that *all other correlation coefficients were generated by chance*. It estimates this probability using the Z-score:

$$P(cc_{ik}) \propto exp(-(Zres_{ik})^2/2) \tag{2.18}$$

Substituting and simplifying, the probability of residue type i at position k is:

$$P_{ik} \leftarrow P_{i0} \cdot \frac{exp\big((Z_{ik}^{res})^2/2\big)}{\sum_l P_{l0} \cdot exp\big((Z_{lk}^{res})^2/2\big)} \tag{2.19}$$

Finally, given these probabilities, RESOLVE finds the alignment of sequence to structure that maximizes the product of probabilities. The final step is, given an alignment-determined residue type at each position, placing the sidechain conformation of the correct type with the highest correlation coefficient Z-score. RESOLVE's final predicted structure on the running example is illustrated in Figure 2.13c.

2.2.2.5 Discussion

Unlike ARP/wARP, RESOLVE uses higher-order constructs than individual atoms in tracing a protein's chain. Searching for helices and strands in the map lets RESOLVE produce accurate traces in poor-quality maps, in which individual atoms are not visible. This method is also widely used by crystallographers. RESOLVE has been successfully used to provide a full or partial interpretation at maps with as poor as 3.5Å resolution. Because the method is based on heuristics, when map quality gets worse, the heuristics fail and the interpretation is poor. Typically, the tripeptide extension is the first heuristic to fail, resulting in RESOLVE traces with many disconnected secondary structure elements. In poor maps, RESOLVE may have difficulty identifying sidechain types. However, RESOLVE is able to successfully use background knowledge from structural biology in order to improve interpretation in poor-quality maps.

Algorithm 2.4 RESOLVE's sidechain-placement algorithm.

input: map $\mathbf{M}$, backbone trace $\mathbf{X} = \{\vec{x}_i\}$ (including C_β's),
sidechain library $\mathbf{F}$, sequence seq
output: all-atom protein model

`// align each sidechain in the library to each predicted C`α
for *each sidechain* $f_j \in \mathbf{F}$ **do**
  for *each C*α $\vec{x}_k \in \mathbf{X}$ **do**
    `// compute correlation, see Equation 2.13`
    $cc_{jk} \leftarrow RSCC(\mathbf{M}(\vec{x}_k), f_j))$
    $Z_{jk} \leftarrow (cc_{jk} - \langle cc_j \rangle)/\sigma_j$
  end
end

`// Estimate probabilities` P_{ik} `that residue type` i `is at position` k
for *each residue type* i **do**
  $P_{i0} \leftarrow$ *a priori* distribution of residue type i
  $Z_{ik} \leftarrow \max_{\text{fragment } j \text{ of type } i} Z_{jk}$
  for *each alpha carbon* $\vec{x}_k \in \mathbf{X}$ **do**
    $P_{ik} \leftarrow P_{i0} \cdot exp(Z_{ik}^2/2) \,/\, \sum_l P_{l0} \cdot exp(Z_{lk}^2/2)$
  end
end

`// Align trace to sequence, place individual atoms`
align seq to chains maximizing product of P_{ik}'s
if *(good alignment exists)* **then**
  place best-fit sidechain of alignment-determined type at each position
else
  return backbone-only model
end

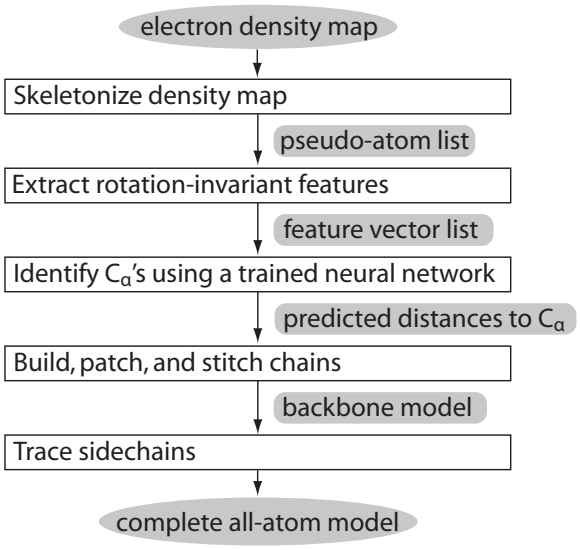


Figure 2.14 A flowchart of TEXTAL.

2.2.3 TEXTAL

TEXTAL is another method for poor-quality density map interpretation, developed by Ioerger *et al*. Much like RESOLVE, TEXTAL seeks to expand the limit of interpretable density maps to those with medium to low resolution (2 to 3Å). TEXTAL uses computer vision and machine learning techniques to match patterns of density in an unsolved map against a database of known density patterns. Matching two regions of density, however, requires an expensive rotational alignment. To deal with this, TEXTAL uses of a set of rotationally invariant numerical features to characterize regions of density. The electron-density in a region around each point in the map is converted to a vector of 76 features sampled several radii that remain constant as the region rotations. The vector consists of descriptors of density, moments of inertia, statistical variation, and geometry.

TEXTAL's algorithm – outlined in Figure 2.14 – attempts to mimic the process by which crystallographers identify protein structures. The first step is to identify the backbone – the location of each Cα – of the protein. Tracing the backbone is done by a subroutine called CAPRA (C-Alpha Pattern Recognition Algorithm), which uses a neural network – a nonlinear function approximator – to estimate the distance from each point in the map to the nearest Cα in the protein. These putative Cα locations are then sent into the second part of the algorithm, LOOKUP, which identifies the sidechains corresponding to each Cα's. Finally, postprocessing corrects flipped segments as well as improving sidechain identification by aligning the model to the input sequence.

2.2.3.1 Feature extraction

A key component of TEXTAL is its extraction of a set of numerical features from a region of density. These numerical features allow rapid identification of similar regions from different

(solved) maps. A key aspect of TEXTAL's feature set is that these features are *invariant to rotations of the region*. This eliminates the need for an expensive rotational search for each fragment; additionally, a discrete rotational search is likely to underestimate some match scores if the true rotation falls between rotational samples.

TEXTAL uses 76 such numerical features to describe a region of density in a map. These features include 19 rotationally invariant features at each of four different radii: 3, 4, 5 and 6Å. The use of multiple radii is critical for differentiation between sidechains: large residues often look similar at smaller radii but greatly differ at 6Å, while small amino acids may have no density in the outer radii and thus are only differentiated at 3 and 4Å.

These 19 rotation-invariant features fall into four basic classes, shown in Table 2.1. The first class describes statistical properties of these neighborhoods of density, treating density values in the neighborhood as a probability distribution. These features include mean, standard deviation, skewness, and kurtosis, the last two of which provide descriptions of the lopsidedness and peakedness of the distribution of density values. The second class of features is really just a single feature: the distance from the center of mass to the center of the neighborhood.

A third class of descriptors includes moments of inertia (MOI), which provides six features describing how elliptical is the density distribution. Moments of inertia are calculated as the Eigenvectors of the inertia matrix I:

$$I = \sum_i \left(\rho_i \left| \begin{array}{ccc} y_i^2 + z_i^2 & -x_i y_i & -x_i z_i \\ -x_i y_i & x_i^2 + z_i^2 & -y_i z_i \\ -x_i z_i & -y_i z_i & x_i^2 + y_i^2 \end{array} \right| \right)$$

Above, ρ_i is the density at point $\langle x_i, y_i, z_i \rangle$. As a rotation-invariant description, TEXTAL only considers the moments and the ratios between moments, not the axes themselves (the Eigenvectors of the inertia matrix).

The final class of features represent higher-level geometrical descriptors of the region. Three "spokes of maximal density" are extended from the center of the region, spaced $> 75^\circ$ apart. These aim to approximate the direction of the backbone N-terminus, the backbone C-terminus, and the sidechain. Rotation-invariant features derived from these spokes include the minimum, median and maximum angle, the *sum* of the angles, the density sum along each spoke, and the area of the triangle formed by connecting the end points of the spokes.

2.2.3.2 Backbone tracing

CAPRA, a component of TEXTAL, produces the initial Cα trace. CAPRA constructs a backbone chain using a neural network. An overview of the process is illustrated in Algorithm 2.5.

In order to accurately compare maps, CAPRA begins by first normalizing density values in the map, ensures feature values from different maps are comparable. Next, CAPRA *skeletonizes* the map, creating a trace of *pseudo-atoms* along the medial axis (or *skeleton*) of some density map contour. Figure 2.15a illustrates this skeletonization. This trace is a very crude approximation of the backbone, and may traverse the side-chains or form multiple distinct chains.

Table 2.1 The rotation-invariant features used by TEXTAL.

Class	*Description*	*Quantity*
Statistical Features of Density		4
	average, standard deviation, skewness, kurtosis	
Center of Mass		1
	distance from center of sphere to center of mass	
Moments of Inertia		6
	magnitude of primary, secondary, tertiary moments; ratios between these moments	
Spokes/Geometry of Density		8
	angles between three "spokes of maximal density" sum of angles, radial densities of each spoke, area of triangle formed by spokes	

Algorithm 2.5 TEXTAL's CAPRA subroutine for calculating the initial backbone trace.

input: electron-density map $\mathbf{M}$
output: putative backbone trace $\mathbf{X} = \{\vec{x}_i\}$

`// consider all skeleton points as potential Cα's`
$\mathbf{pseudoAtoms} \leftarrow skeletonize(\mathbf{M})$

`// use neural network to predict dist-to-Cα for each skeleton point`
for $p_i \in \mathbf{pseudoAtoms}$ **do**
  $\mathbf{F} \leftarrow$ rotation-invariant features in a neighborhood around p_i
  distance-to-$C_\alpha \leftarrow neuralNetwork(\mathbf{F})$
end
`// using a heuristic, assemble chains from predicted distances`
$\mathbf{X} \leftarrow$ construct chain using predicted distances-to-Cα

A feed-forward neural network – a nonlinear function approximator used for both classification and regression – is trained to learn which pseudo-atoms correspond to actual Cα's. Training is done using the *backpropagation* algorithm, which learns network parameters using (stochastic) gradient ascent in parameter space.

Specifically, the network is trained on a set of previously solved maps to predict the *distance* of each pseudo-atom to the nearest Cα. The rotation-invariant features are inputs to the network; a single output node estimates the distance to the closest Cα. A hidden layer of 20 sigmoidal units fully connects input and output layers.

Given a predicted distance-to-Cα for each pseudo-atom, CAPRA uses a greedy trace to find a linear chain linking Cα's together. Further post-processing has been added to improve performance of CAPRA, including mainchain refinement and patching missing pieces. CAPRA's output on the sample map at 2.5Å resolution, is shown in Figure 2.15b.

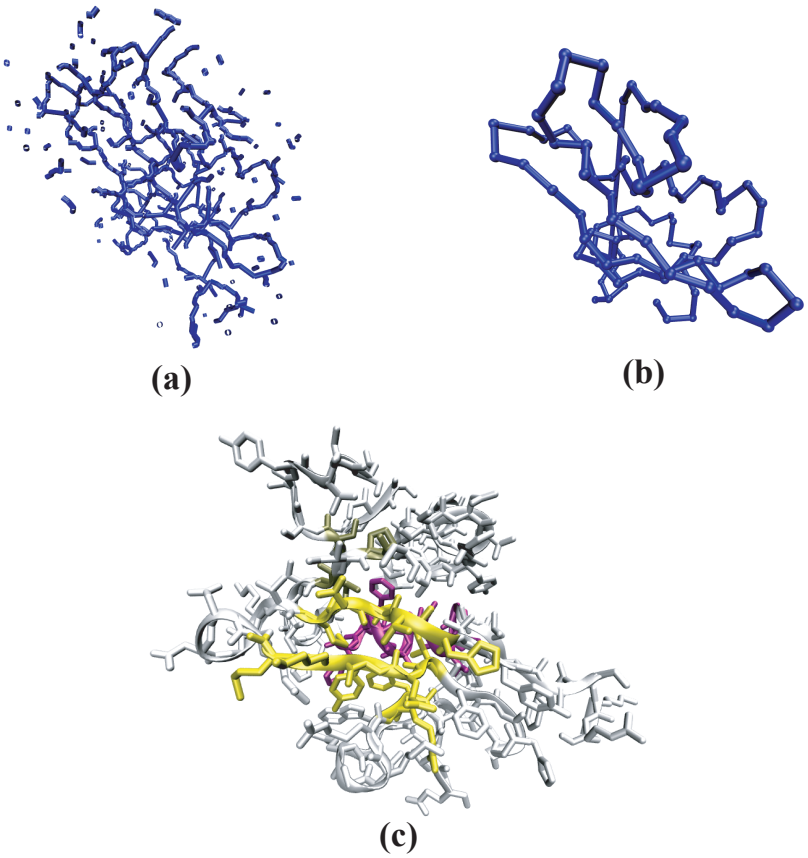


Figure 2.15 Intermediate steps in TEXTAL's structure determination: (a) the skeletonized density map, which crudely approximates the protein backbone, (b) a backbone trace, which TEXTAL builds by determining Cα's from the skeleton points, and (c) the final predicted structure.

2.2.3.3 Sidechain placement

After CAPRA returns its predicted backbone trace, TEXTAL must next identify the residue type associated with each Cα. This identification is performed by a subroutine LOOKUP. Algorithm 2.6 shows a pseudocode overview of LOOKUP. Essentially, the subroutine compares the density around each Cα to a database of solved maps to identify the residue type. LOOKUP uses TEXTAL's rotation-invariant features, and builds a database of feature vectors corresponding to Cα's in solved maps. To determine the residue type of an unknown region of density, LOOKUP finds the *nearest neighbors* in the database, using weighted Euclidian distance:

$$D(\rho_1, \rho_2) = \Big[\sum_i \lambda_i \cdot \big(F_i(\rho_1) - F_i(\rho_2)\big)^2 \Big]^{1/2} \tag{2.20}$$

Algorithm 2.6 TEXTAL's LOOKUP subroutine for placing sidechains.

input: electron-density map $\mathbf{M}$, backbone trace $\mathbf{X} = \{x_i\}$
output: all-atom protein model

```
// place a sidechain on each putative Cα
for x_i ∈ X do
    // find k sidechain conformations matching density at x_i
    //     use rotation-invariant representation
    F ← rotation-invariant features in a neighborhood around x_i
    N ← k examples in DB minimizing weighted Euclidean distance

    // compute correlation to each neighbor
    for n_i ∈ N do
        n_i' ← optimal superposition of n_i into map at x_i
        score_i ← RSCC(n_i, M(x_i))
    end
    // add best-matching conformatino to model
    Choose n_i' maximizing score_i
    Add individual atoms of n_i' to model
end
```

Above, F_i refers to the ith rotation-invariant feature, while ρ_1 and ρ_2 are two regions of density. Feature weights λ_i are learned to *maximize* similarity between matching regions and *minimize* similarity between non-matching regions (ground truth is the optimally-aligned $RSCC$, see Equation 2.13). TEXTAL sets weights using the SLIDER algorithm [39] which considers features pairwise to determine a good set of weights.

Since information is lost when representing a region as a rotation-invariant feature vector, the nearest neighbor in the database does not always correspond to the best-matching region. Therefore, LOOKUP lookup considers the k closest regions in the database, and performs a time-consuming rotational alignment on each of these. Ideally, LOOKUP wants to find the rotation of each of the k regions to maximize the correlation coefficient. It quickly approximates this optimal rotation and translation by aligning moments of inertia between template density region ρ_1 and target density region ρ_2, computing the real-space correlation at this alignment. LOOKUP selects the candidate (of the k choices) with highest correlation.

2.2.3.4 Post-processing routines

Since each residue's atoms are copied from previous examples and glued together in the density map, the model produced by LOOKUP may contain some physically infeasible bond lengths, bond angles, or $\phi - \psi$ torsion angles. TEXTAL's final step is improving the model using a few simple post-processing heuristics. First, LOOKUP often reverses the backbone direction of a residue;

TEXTAL's post-processing makes sure that all chains are oriented in a consistent direction. Refinement, like that of ARP/wARP, corrects improper bond lengths and bond angles, iteratively moving individual atoms to fit the density map better. Finally, TEXTAL takes into account the target protein's sequence to fix mismatched residues.

TEXTAL makes use of a provided sequence by aligning the map-determined model sequence to the provided input sequence, using a Smith-Waterman dynamic-programming alignment [103]. If there is agreement between the sequences above some threshold, then a second LOOKUP pass corrects residues where the alignment disagrees. In this second pass, LOOKUP is restricted to only consider residues of the type indicated by the sequence alignment. Like RESOLVE, TEXTAL's end result is a complete all-atom protein model, illustrated for our example map in Figure 2.15c.

2.2.3.5 Discussion

TEXTAL – like RESOLVE – uses higher-order constructs than atoms in order to successfully solve low-quality maps. In practice, TEXTAL works well on maps at around 3Å resolution. TEXTAL's key contribution is the use of rotation-invariant features to recognize patterns in the map. This feature representation allows accurate Cα identification using a neural network; it also does well at classification of amino-acid type. TEXTAL tends to do better than RESOLVE at sidechain identification due to this feature set. One key shortcoming, however, is limiting the initial backbone trace to skeleton points in the density map. In very poor maps, skeletonization inaccurately traces the backbone. This is in part responsible for TEXTAL's failure in maps worse than 3Å. However, TEXTAL has successfully employed previously solved structures and domain knowledge from structural biology to produce an accurate map interpretation.

2.2.4 Summary of approaches

A key step in determining protein structures is interpreting electron-density maps. In this area, bioinformatics has played a key role. This chapter describes how three different algorithms have approached the problem of electron-density map interpretation:

- The WARPNTRACE procedure in ARP/wARP was the first method developed for automatic density-map interpretation. Today, it is still the most widely used method by crystallographers. ARP/wARP uses an "atom-level" technique in which free atoms are first placed into the density map, free atoms are next linked into chains introducing constraints, and finally, the combined model is refined to better explain the map. The method iterates through these three phases, at each iteration using the partial model to improve the map. Because it works at the level of individual atoms, it requires 2.3Å or better map resolution.

- RESOLVE is a method that searches for higher-level constructs – amino acids and secondary-structure elements – as opposed to ARP/wARP's atom-level method. Consequently, it produces accurate traces in poor-quality (around 3Å) electron-density maps, unsolvable by ARP/wARP. It, too, is widely used by crystallographers. RESOLVE begins by matching short secondary-structure fragments to the maps, then uses a large fragment library to extend

these matches. Finally, overlapping matches are merged in a greedy fashion, and sidechains are traced. Incorporating structural domain knowledge is key to this method's success.

- TEXTAL also accurately interprets medium to low resolution (2 to 3Å), using residue-level matching. It represents regions of density as a vector of rotation-invariant features. This alternate representation serves several purposes. It is used to learn a classifier to identify Cα's, and it is also used to recognize the residue type of each putative Cα through a database comparison. The classifier is also very well suited to identifying the residue type in a given region, making it more accurate than RESOLVE in this respect. Additionally, TEXTAL's rotation-invariant representation enables fast matching of regions of density. This allows TEXTAL to easily make use of previously solved structures (via the information retained by the trained neural network) in its map interpretation, providing accurate traces even in poor-quality maps.

2.3 Discussion

This chapter gave a brief background on important structural biology concepts, as well as an introduction to probabilistic reasoning and inference. I described the problem of electron-density map interpretation, and introduced several other algorithms used in automatic interpretation. I also took a detailed look at three widely used methods for automatic density-map interpretation.

The remainder of my thesis describes a novel approach to automatic density-map interpretation. Using probabilistic reasoning, my approach will find the most likely protein structure for *some particular sequence* in a given density map. This contrasts with the approaches presented in this chapter, which separate the tasks of tracing the protein backbone and aligning the predicted backbone to the given input sequence. The result is a method that – at poor map resolutions – has better sidechain identification and more complete, less fragmented models than the approaches presented above.

Chapter 3

A Probabilistic Approach to Protein-Backbone Tracing

In this chapter, I describe ACMI[1] (Automatic Crystallographic Map Interpreter), an algorithm that automates backbone tracing in electron-density maps. This chapter describes two main components: ACMI-FF, a *local matching* component that locates individual amino acids in the density map, and ACMI-BP, a *global constraint* component that uses prior knowledge of the protein's structure to eliminating false positives from the local matching. ACMI-BP implements an efficient inference algorithm that can infer the protein's backbone layout in an electron-density map. Throughout this chapter, "ACMI" will refer to the sequential application of these two components, ACMI-FF and ACMI-BP.

My model is probabilistic: throughout the interpretation it represents each residue as a probability distribution over the electron-density map. This property – not being constrained to force each residue to a single location – is advantageous as it naturally handles noise in the map and disordered regions in the protein. An earlier version of this chapter was previously published [27].

3.1 Overview of the algorithm

A high-level overview of ACMI's two main components is illustrated in Algorithm 3.1. ACMI includes a local-matching component, ACMI-FF (for *fast-Fourier matching*) where individual residues are probabilistically located in the map, independent of all other residues, and a global-constraint component, where the backbone chain is built up, also probabilistically, from the local matches, taking into account the chemical laws governing the physical structure of proteins.

The local-matching component of my algorithm makes use of a library of existing sequence-specific 5-mer (5-amino-acid long) templates. That is, when searching for an individual residue, I actually look for all common conformations of the 5-mer centered at that residue. The local search has high sensitivity, usually matching well to the residue's correct location. However, it suffers from low specificity, producing a significant number of false positives.

ACMI's global-constraint component, ACMI-BP (for *belief propagation*) probabilistically refines these local search results to take into account prior knowledge of protein structure. Using

[1]Throughout the thesis, I will use the notation ACMI-*XX* to refer to individual components of ACMI (e.g., ACMI-FF, ACMI-SH, ACMI-BP, and ACMI-PF). I will refer to the entire map-interpretation pipeline using ACMI, making sure to specify which individual components I am using.

Algorithm 3.1 ACMI's algorithm for inferring protein Cα locations. Vector symbols are dropped for clarity.

input: Sequence seq and electron-density map $\mathbf{M}$
output: Putative backbone trace $\mathbf{U}^* = \{u_i^*\}$
// The variable u_i^* describes the position of amino-acid i's Cα
// and orientation

foreach *amino-acid* i **do**
  $P(\mathbf{M}|u_i) \leftarrow$ doLocalMatch$(seq_i, \mathbf{M})$
end
$P(\mathbf{U}) \leftarrow$ enforceGlobalConstraints$(seq, \{P(\mathbf{M}|u_i)\})$
$\mathbf{U}^* \leftarrow \{u_i^* | \forall i u_i^* = \arg\max_{u_i} P(u_i)\}$

// Search the map for a set of templates corresponding to each
// amino acid in the protein
procedure doLocalMatch$(seq, \mathbf{M})$
  input: 5-mer sequence seq_i and electron-density map $\mathbf{M}$
  output: Prob. distribution $P(\mathbf{M}|u_i)$ of each residue i over map $\mathbf{M}$

  $\mathbf{fragments} \leftarrow$ extract all conformations of seq_i from PDB
  $(\mathbf{centroids}, \mathbf{weights}) \leftarrow$ cluster $\mathbf{fragments}$ using all-atom RMSd
  foreach $centroid_j \in \mathbf{centroids}$ **do**
    $t_j(\vec{x}, \Theta) \leftarrow$ mismatch between density map at $\vec{x}$ and $centroid_j$ rotated by Θ
    $P_j(\vec{x}, \Theta) \leftarrow$ use tuning set to convert t_j's to probabilities
  end
  $P(\vec{x}, \Theta) \leftarrow \sum_j weight_j \cdot P_j(\vec{x}, \Theta)$

// Given local-match probabilities and structural constraints,
// infer each amino acid's location
procedure enforceGlobalConstraints$(seq, \{P(\mathbf{M}|u_i)\})$
  input: Individual amino-acid *prior distributions*
  output: Marginal (posterior) probabilities $\hat{p}_i(\vec{u_i})$ under structural constraints

  $\mathcal{G} \leftarrow$ Construct undirected graph from protein sequence
    Vertices $\mathcal{V}$ model Cα positions
    Edges $\mathcal{E}$ enforce (pairwise) structural constraints
  $\hat{p}_i(\vec{u_i}) \leftarrow$ Belief-propagation inference on $\mathcal{G}$

this prior knowledge, it adjusts the local-match probabilities based on the local-match probabilities of other residues. It produces a physically feasible interpretation that maximizes the probabilities from the local matching.

ACMI-BP models this physical feasibility with a pairwise Markov field [36], which represents the probability of a conformation as the product of *potential functions* between pairs of residues. This pairwise potential is analogous to the pairwise potential-energy calculations used in molecular dynamics [62] (although my model does not optimize physical energy but rather statistical "energy").

3.2 Local matching

I developed ACMI-FF to locate individual protein residues in an electron-density map. In the poor-quality maps for which ACMI-FF is designed, simple atom-based refinement methods [94] perform poorly. Empirically (reported in the respective method's publications), methods using rotamer searching [111], skeletonization [40], or critical points [73] also perform poorly in these low-resolution maps. The methods that have had the most success in low-resolution maps are those based upon finding large fragments of protein electron-density [17]. Thus, I use sequence-specific 5-mer search to locate individual residues in the electron-density map.

My method is divided into two basic parts, illustrated in Figures 3.1 and 3.2. First, I use previously solved structures from the Protein Data Bank [3] to construct a basis set of sequence-specific 5-mer templates. I then perform a 6D (rotation + translation) search in the map for each of the 5-mers in my basis set. The output of this local search is – for each residue – an estimated probability distribution of that residue's presence over the unit cell. The remainder of this section presents one approach to searching for a set of density templates in the electron-density map; Chapter 4 presents an improved approach.

3.2.1 Constructing a sequence-specific 5-mer basis set

ACMI-FF begins this step – illustrated in Figure 3.1 – by walking along the one-dimensional protein sequence, considering a 5-mer centered at each residue. Given this 5-mer, ACMI-FF searches a non-redundant subset of the PDB [115] (restricted to have less than 25% sequence similarity) for three-dimensional instances of that 5-mer. If there are less than 50 such instances then ACMI-FF considers structures with *similar* 5-amino-acid sequences. Similarity is determined using PAM distance [58], an amino-acid distance metric based on the probability that a particular amino acid will mutate into another. ACMI-FF considers increasing the PAM-distance threshold until there are at least 50 structures with PAM distance to the target less than or equal to the threshold.

It is infeasible to search for all 50+ conformations in the electron-density map, so alternately, ACMI-FF clusters the structures and represents each cluster as a centroid fragment and a weight. When clustering the fragments, I use rotationally-aligned, all-atom RMS deviation between fragments as a distance metric (quickly computed as an optimization problem [63]). Clustering uses

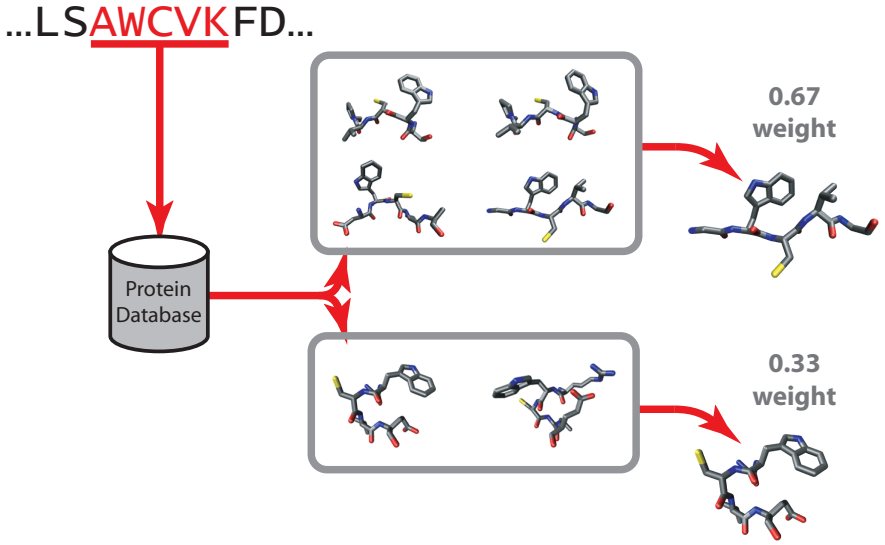


Figure 3.1 My 5-mer clustering process. Walking along the given amino-acid sequence, I consider a 5-mer centered at each position. I search the database for instances of that 5-mer, and cluster them. Finally, I extract a representative member from each cluster. This characterizes the conformational space of the 5-mer sequence.

complete-linkage hierarchical clustering [57], limiting clusters to have a maximum diameter of 3Å. That is – in any cluster – no two structures have an RMSd greater than 3Å.

Any cluster with under 10% representation is thrown out (to limit CPU time in the next step); in all remaining clusters I find a centroid (i.e. representative) fragment. Each remaining cluster is assigned a weight proportional to the number of structures the cluster contains. Depending on the "sequence structural entropy" of the 5-mer [50], ACMI-FF typically produces anywhere from 1 to 7 clusters (and resultant centroid fragments).

The cluster centroids and the weights determined by ACMI-FF represent these "basis templates" (or "basis fragments") for each specific 5-mer sequence. Using fragments of length five is my way of balancing the trade-off between template size and template specificity. Larger fragments are preferred for recognition in poor-quality maps, but larger fragments have lower representation in the set of already-solved structures. ACMI-FF's non-redundant PDB subset contains about 20% of the 3.2×10^6 possible 5-mers.

3.2.2 Searching the map for 5-mer template structures

Once clustering is complete and the cluster centroids have been extracted, ACMI-FF searches for instances of the centroids in the electron-density map. This process is illustrated in Figure 3.2. Given a fragment and a target resolution, I build a map corresponding to what I would expect to

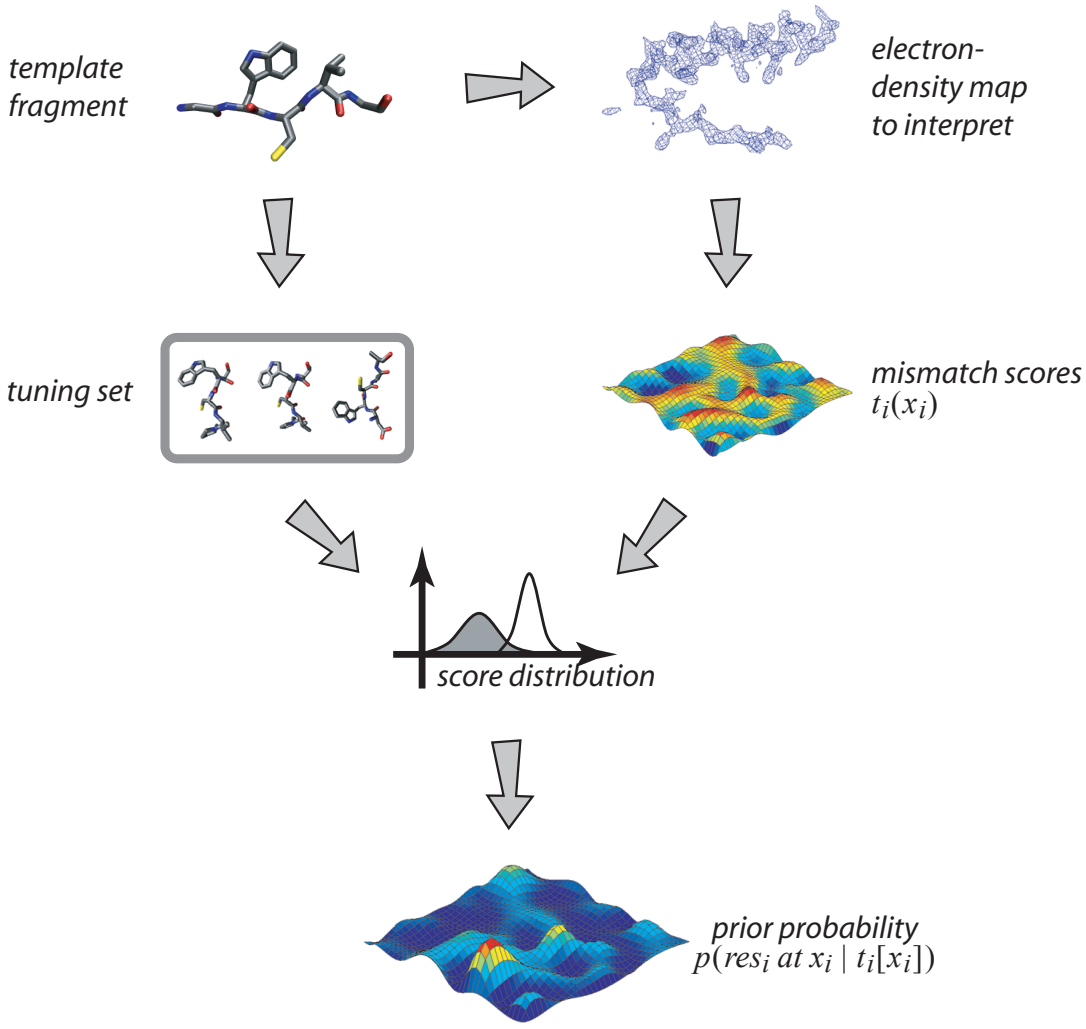


Figure 3.2 An overview of the 5-mer template matching process. After I have extracted a representative set of 5-mers for each residue i, I perform a 6D (rotation + translation) search for the fragment in the density map. By also matching the fragment to a tuning set of known structures, I use Bayes' rule (see Equation 3.3) to estimate the probability distribution of the Cα's location in the density map.

see (a description of this process is given in Section 2.1.2). Then, at each map location, I compute the mean squared electron-density difference $t(\vec{x})$ between the map and the fragment. I compute this difference over all points $\vec{x} = (x_i, y_i, z_i)^{\mathrm{T}}$ in the electron-density map in a fragment-shaped neighborhood around $\vec{x}$,

$$t(\vec{x}) = \sum_y \varepsilon_f(\vec{y}) \Big(\rho'_f(\vec{y}) - \frac{1}{\sigma_\rho(\vec{x})} [\rho(\vec{y} - \vec{x}) - \bar{\rho}(\vec{x})] \Big)^2. \tag{3.1}$$

Here, $\rho(\vec{x})$ is the density map in which I am searching (that is, ρ is the electron-density function), while $\rho'_f(\vec{y})$ is the *standardized* (that is normalized to have mean $\mu = 0$ and variance $\sigma^2 = 1$) fragment electron-density. $\varepsilon_f(\vec{y})$ is a masking function that is nonzero only for points in this fragment-shaped region around $\vec{y}$, and $\bar{\rho}(\vec{x})$ and $\sigma_\rho(\vec{x})$ are scaling functions, ensuring that the masked region of the map and template have the same mean and standard deviation. Specifically, the function $\bar{\rho}_\varepsilon(\vec{x})$ is the average density over the mask ε_f around the point $\vec{x}$, while $\sigma_\varepsilon(\vec{x})$ scales the standard deviation of the density map around $\vec{x}$,

$$\sigma^2_\varepsilon(\vec{x}) = \frac{\sum_y \varepsilon_f(\vec{y}) [\rho(\vec{y} - \vec{x}) - \bar{\rho}(\vec{y})]^2}{\sum_y \varepsilon_f(\vec{y})}. \tag{3.2}$$

ACMI-FF needs to perform the fragment search as a 6D search over all rotations plus all translations; fortunately, I compute this quickly over the whole map (at a single rotation) using FFTs [16]. Additionally, at each position ACMI-FF stores the best-matching 5-mer fragment, and the corresponding rotation, for later use.

The electron-density difference function $t(\vec{x})$ is a good measure of similarity between regions of density, but I need a way to convert these scores into probability distributions. That is, I want the probability $P(\vec{x_i}|score_i)$ that a specific 5-mer cluster i is present at location $\vec{x_i}$, given match score $score_i$. ACMI-FF computes this using a tuning set and the application of Bayes' rule. Bayes' rule gives us this probability:

$$P(\vec{x_i}|score_i) = P(score_i|\vec{x_i}) \times \frac{P(\vec{x_i})}{P(score_i)}. \tag{3.3}$$

. The terms on the right-hand side are computed or estimated as follows. The probability distribution of match scores over the map, $P(score_i)$, is derived from the actual distribution of match scores over the (unsolved) map. The prior probability on a residue's location over the map, $P(x_i)$, is simply a normalization term: I already know (by knowing the protein's sequence) the number of copies of the 5-mer in the electron-density map, and so I normalize probabilities over the map to reflect this value.

However, the first term – the distribution of scores when a 5-mer matches the map – is trickier to compute. ACMI-FF estimates this term using a *tuning set* derived from different protein structures from the PDB. This tuning set contains other instances from the same 5-mer cluster for which I am searching.

I match each centroid's density with each tuneset example's density to estimate the distribution of scores given a 5-mer match. I compute the squared-density difference between: (a) a centroid and (b) the optimally rotated tuneset example. I assume this distribution is normally distributed, and choose the maximum-likelyhood parameters μ_{match} and σ^2_{match}.

3.2.3 Additional sources of local information

When manually solving an electron-density map, a crystallographer uses all of the information available. This includes not just the electron-density map and the protein primary sequence, but also other sources of "soft" information, such as distant structural analogues, predicted secondary structures and the locations of heavy atoms from phasing experiments.

In particular, one commonly used experiment to recover phase information for an electron-density map is *multi-wavelength anomalous diffraction* (MAD) [44], which uses the property that certain heavy atoms tend to absorb X-rays at a particular wavelength. Using the scattering patterns produced by different X-ray wavelengths, one can determine the locations of these heavy atoms, and use these atoms to determine initial map phases.

Selenium is one such heavy atoms. When solving protein structures that have sulfur-containing methionine residues, one common source of heavy atoms is to "replace" the sulfur in methionine with selenium. The protein is grown using the selenium-containing selenomethionine. Using MAD, the locations of these selenium atoms must be determined before the density map is calculated.

When solving a density map phased in this manner, a crystallographer is provided the locations of each of the seleniums in the map; each of the protein's methionines must be placed coincident with these locations. A crystallographer can use this knowledge to guide model construction. However, this must be viewed as a soft constraint, since: (a) not all the seleniums may be found, and (b) there may be false-positive selenium locations.

ACMI-FF takes advantage of this information – when available – by "up-weighting" selenomethionine probabilities around each predicted selenium location (selenomethionine's Cα is typically 3-4.5Å from its selenium), and "down-weighting" all other probabilities. Specifically, I compute a modified probability for each selenomethionine (MSE is the three-letter abbreviation for selenomethionine):

$$\begin{aligned} P_{MSE}(\vec{x_i}|score_i, \textbf{SE-locs}) &= P(score_i, \textbf{SE-locs}|\vec{x_i}) \times \frac{P(\vec{x_i})}{P(score_i, \textbf{SE-locs})} \\ &= P(score_i|\vec{x_i}) \times P(\textbf{SE-locs}|\vec{x_i}) \times \frac{P(\vec{x_i})}{P(score_i) \times P(\textbf{SE-locs})} \end{aligned}$$

The second line of the equation results from assuming that match scores and selenium locations are conditionally independent given the 3D structure. The term $P(\textbf{SE-locs})$ is folded into normalization, leaving only the term $P(\textbf{SE-locs}|\vec{x_i})$ as an additional computation. ACMI-FF calculates this term using empirical data (from the PDB), as well as a *user-provided* "confidence" in the fidelity of provided Cα locations.

Specifically, ACMI-FF learns a potential function $\psi_{MSE}(c, s)$ that gives the probability of observing a selenomethionine alpha-carbon at 3D location c, and the corresponding selenium at s. This function is learned from the non-redundant PDB subset used for ACMI-FF's local matching. The function only depends on the *distance* between c and s, and is modeled using the weighted sum two variable-width Gaussians. I learn a total of 5 parameters (the ratio of weights, two means, and two variances) describing this distribution using expectation-maximization (EM) [21].

The distribution I learn is:

$$\psi_{MSE}(c,s) \propto w_1 \cdot \mathcal{N}(||c-s||;\mu_1,\sigma_1^2) + (1-w_1)\cdot\mathcal{N}(||c-s||;\mu_2,\sigma_2^2) \tag{3.4}$$

The parameters I learn are $w_1 = 0.6$, $\mu_1 = 3.3$Å, $\mu_2 = 4.2$Å, $\sigma_1 = 0.1$Å, and $\sigma_2 = 0.04$Å. This function is normalized to sum to unity over the density map.

Given this potential function, the definition of $P(\textbf{SE-locs}|\vec{x_i})$ is straightforward:

$$P(\textbf{SE-locs}|\vec{x_i}) = \frac{P_0}{||SE-locs||}\cdot\sum_{s_j\in\textbf{SE-locs}}\psi_{MSE}(x_i,s_j) + \frac{1-P_0}{V} \tag{3.5}$$

The variable V is the density-map volume (i.e., the number of grid points). The term P_0 is a "confidence term"; that is, how confident the crystallographer is in the correctness and completeness of the list of selenium locations. Since this value is highly variable from map to map – and the crystallographer may have some insight to its value based on results from phasing experiments – I make it a user-provided parameter. All experiments using this term (in my thesis, only the experiments in Chapter 5 make use of it) use $P_0 = 0.9$.

3.2.4 Discussion

At the end of the local-matching procedure, ACMI-FF has computed – for each residue – a probability distribution over all rotations and translations. That is, for each location $\vec{x}$ in the density map, and each rotation Θ, I have a probability that each Cα is positioned at that location/rotation. The remainder of the chapter describes how my algorithm uses *prior knowledge about the structure of the protein* to estimate the most probable backbone trace given these probability distributions.

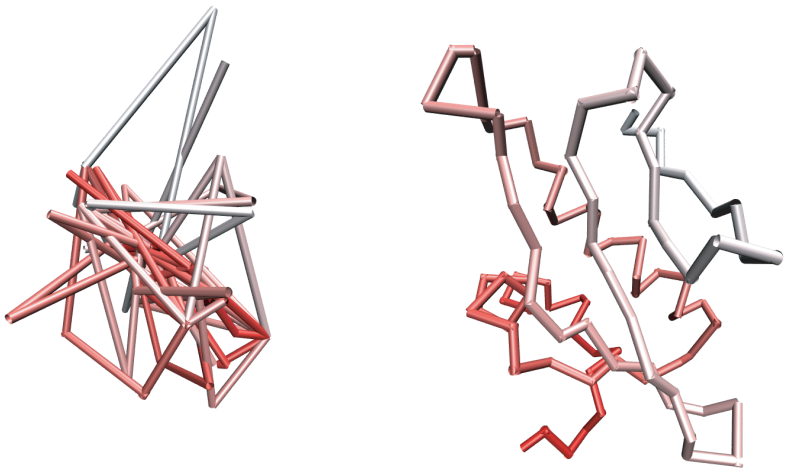
Run times for the local matching are significant: for each fragment I have to search ≈ 1900 rotations (20-degree discretization) over the entire electron-density map. The total compute time is on the order of CPU-weeks; however, 5-mer matching is trivially parallelized [112], since each 5-mer can be processed separately and independently of one another (further straightforward parallelization is possible).

3.3 Global constraints

In Section 3.2, I compute – for each residue i – a probability distribution over every position $\vec{x}$ and rotation Θ in the unit cell. This rotational variable Θ includes not only the three Euler angles (α,β,γ), but also an internal "bend" angle δ, defined as the angle formed by three consecutive Cα's. For notational convenience, I will refer to this seven-dimensional variable as $\vec{u} = (\vec{x},\Theta)$.

This seventh dimension δ is important because – in this second phase – I want to answer the question: Given one Cα is located at a point $\vec{u_i}$, where can I find the adjacent Cα's? Without providing this "bend" angle (which may range from 90° to 180°), the answer is ambiguous.

Alternately, one can think of this probability distribution in a *generative* sense, as the probability that this map $\mathbf{M}$ was generated by amino-acid i at location and rotation $\vec{u_i}$, that is, as the



(a) (b)

Figure 3.3 Two possible backbone traces. The trace (a) maximizes the product of 5-mer match probabilities; however, the resultant protein is physically impossible. I would prefer trace (b) with non-optimal local-match probabilities, but which corresponds to a physically realistic structure. Shading shows progression along the backbone.

probability $P(\mathbf{M}|\vec{u_i})$. One could presumably select, for each amino-acid i, the $\vec{u_i}$ that maximizes this probability. However, to do so would not take into account that amino acids are not placed independently, but their locations are in fact highly correlated due to physical-chemical constraints.

Figure 3.3 illustrates this idea. Independently choosing locations $\vec{u_i}$ to maximize local match scores gives a model that looks like Figure 3.3(a). A user would much prefer the model in Figure 3.3, where *not every individual* $\vec{u_i}$ *is maximized*, but the overall model corresponds to a physically realistic protein structure.

I somehow need to account for the structural probability on the model. That is, I needs to ensure that the proposed structure is a *physically feasible protein molecule*. What I ultimately want to find – given map $\mathbf{M}$ – is the configuration of all residues $\mathbf{U}^* = \{\vec{u_1^*}, \ldots, \vec{u_N^*}\}$, such that

$$\begin{aligned} \mathbf{U}^* &= \arg\max_{\mathbf{U}} P(\mathbf{U}|\mathbf{M}) \\ &= \arg\max_{\mathbf{U}} \left\{ P(\mathbf{U}) \times \prod_{i=1}^{N} P(\vec{u_i}|\mathbf{M}) \right\} \end{aligned} \tag{3.6}$$

This first term, $P(\mathbf{U})$, encodes the structure's physical feasibility, in which a proposed structure like that of Figure 3.3b would have a much higher probability of configuration than Figure 3.3a. Recall that the second term – computed by ACMI-FF – assumes independent placement of amino acids in the model. The first term accounts for this dependency.

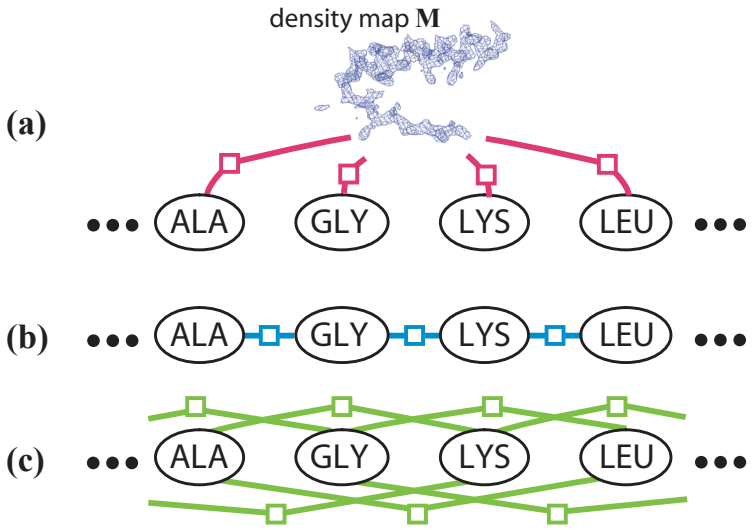


Figure 3.4 The structure of ACMI-BP's graphical model. The joint probability of a conformation of residues is the product of (a) an *observation potential* at each node, (b) an *adjacency potential* between adjacent residues, and (c) an *occupancy potential* between all pairs of non-adjacent residues.

3.3.1 Pairwise Markov-field model

To model the "global constraint" probability, ACMI-BP uses an *undirected graphical model* [6]. An undirected graphical model defines the probability distribution of a set of variables on an undirected graph. ACMI-BP uses a particular type of undirected graphical model known as a pairwise Markov field [36], where the probability of a particular setting of the random variables in the graph is the product of *potential functions* associated with vertices and edges in the graph.

ACMI-BP's pairwise Markov-field model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ consists of a set of nodes $i \in \mathcal{V}$ connected by edges $(i, j) \in \mathcal{E}$. Each node in the graph is associated with a (hidden) random variable u_i. The graph is conditioned on observation variables $\mathbf{M}$. Each vertex has a corresponding *observation potential* $\psi_i(\vec{u_i}, \mathbf{M})$, and each edge is associated with a *conformational potential* $\psi_{ij}(\vec{u_i}, \vec{u_j})$. The model represents the full joint probability as the product of these potentials:

$$P(\mathbf{U}|\mathbf{M}) = \prod_{i \in \mathcal{V}} \psi_i(\vec{u_i}|\mathbf{M}) \times \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(\vec{u_i}, \vec{u_j}) \tag{3.7}$$

I am primarily concerned with finding the $\vec{u_i}$'s maximizing this probability, given some density map $\mathbf{M}$.

Figure 3.4 shows how I encode a protein in a Markov-field model. Each node i represents an amino-acid residue in the protein. The value $\vec{u_i}$ for each amino-acid residue consists of seven terms: the 3D Cartesian coordinates $\vec{x_i}$ of the residue's alpha Carbon (Cα), and the four internal rotational parameters Θ.

The *observation potential* $\psi_i(\vec{u_i}|\mathbf{M})$ associated with each residue is proportional to the 5-mer probability $P(\mathbf{M}|u_i)$, as computed in the previous section.

The conformation potentials $\psi_{ij}(\vec{u_i}, \vec{u_j})$, which model the probability of a particular conformation of the residues in the protein, are further divided into two basic types. Following Sudderth *et al.*'s hand-tracking model [106], ACMI-BP defines *adjacency potentials* associated with each edge connecting neighboring residues (Figure 3.4b). These potentials ensure that adjacent residues maintain the proper 3.8Å spacing and the proper Cα–Cα–Cα angle. ACMI-BP also defines *occupancy potentials* between non-adjacent residues (Figure 3.4c), which prevent two residues from occupying the same region in three-dimensional space.

Expanding Equation 3.7, ACMI-BP's full joint probability of some Cα trace is now given as

$$P(\mathbf{U}|\mathbf{M}) = \prod_{\text{amino-acid } i} \psi_i(\vec{u_i}|\mathbf{M}) \times \prod_{\substack{\text{amino-acids } i,j \\ |i-j|=1}} \psi_{adj}(\vec{u_i}, \vec{u_j}) \prod_{\substack{\text{amino-acids } i,j \\ |i-j|>1}} \psi_{occ}(\vec{u_i}, \vec{u_j}) \tag{3.8}$$

Because residues distant on the protein chain are not necessarily distant in space, the graph must be fully connected; that is, *every* pair of residues is joined by an edge in the Markov-field model.

One can think of this as a "scoring function" for evaluating candidate 3D backbone traces. That is, given candidate backbone trace **U'**, which gives a location to every Cα in a protein, Equation 3.8 is used to evaluate the "goodness" of **U'**, given the density map **M** and knowledge of how proteins tend to fold.

3.3.1.1 Adjacency potentials

The adjacency potentials, which connect every adjacent pair of residues, are further broken down into the product of two constraining functions, a distance constraint function and a rotational constraint function:

$$\psi_{adj}(\vec{u_i}, \vec{u_j}) = p_x(||\vec{x_i} - \vec{x_j}||) \times p_\Theta(\vec{u_i}, \vec{u_j}) \tag{3.9}$$

The distance constraint is based on the physical fact that, in proteins, the Cα–Cα distance is a nearly invariant 3.8Å. Thus, this first term p_x takes the form of a tight Gaussian around this ideal value.

The internal parameters Θ model the 3D rotation of each residue and the angle formed by the residue triple centered at residue i. To simplify the definition of p_Θ, I parameterize these four degrees of freedom as two pairs of (θ, ϕ) spherical coordinates: (θ_f, ϕ_f), the most likely direction of the "forward" (C-terminal) residue, and (θ_b, ϕ_b), the most likely direction of the "backward" (N-terminal) residue. Figure 3.5 illustrates this parameterization.

ACMI-BP's local 5-mer matching – in addition to computing the probability at a specific location – also remembers, for each *translation* $\vec{x}$: (a) the best-matching template fragment, and (b) the best-matching rotation of that centroid. Then, *at each location in the map*, I store four values – θ_f^*,ϕ_f^*, θ_b^*, and ϕ_b^* – indicating the direction of the adjacent residues' Cα's, based on the direction of these Cα's in this rotated, best-matching 5-mer.

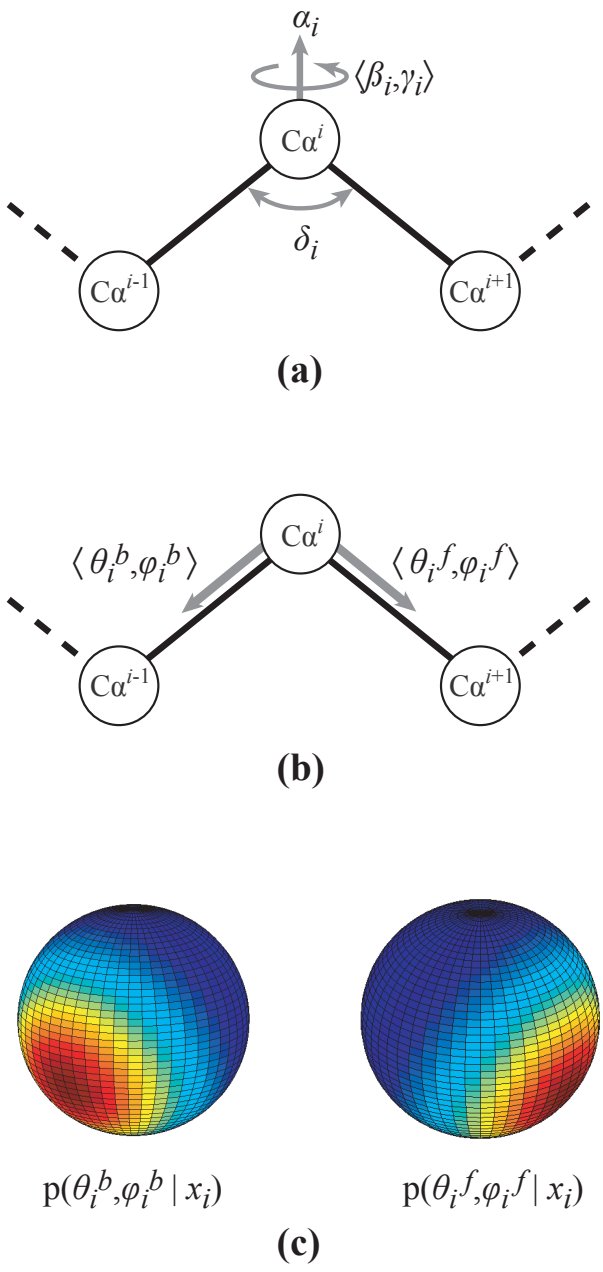


Figure 3.5 An illustration of an amino acid's internal rotational parameters. ACMI-BP parameterizes (a) each amino acid's four degrees of freedom – the three rotational parameters and the "bend" formed by three consecutive Cα's – as (b) the direction of the C-terminal and N-terminal adjacent amino acids. (c) My algorithm assumes that the probability distribution is distributed as a Gaussian in this 4D space.

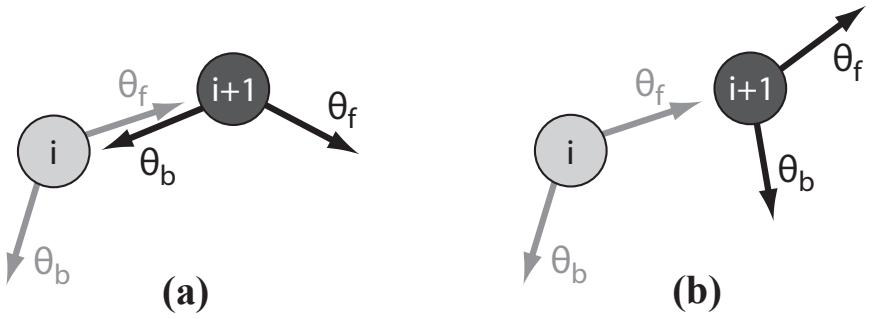


Figure 3.6 The angular constraint function p_Θ ensures the probability of (a) a model where (θ_f^*, ϕ_f^*) of amino-acid i and (θ_b^*, ϕ_b^*) of amino-acid $i+1$ point at each other is *greater* than (b) a model where they do not.

Another way to think about this is that at each location $\vec{x}$ in the map, alpha-carbon i says "if I am at location $\vec{x}$, I think alpha-carbon $i+1$ is most likely in the direction (θ_f^*, ϕ_f^*), and alpha-carbon $i-1$ is most likely in the direction (θ_b^*, ϕ_b^*)". My algorithm assumes that the probability distribution in rotational space (that is, in Θ-space) is distributed as a four-dimensional Gaussian around these stored values, plus some uniform prior probability (justification for this appears in Section B.1). I add a uniform prior (typically 10% of the probability mass) to account for errors introduced by estimating this distribution in Θ-space as a single Gaussian.

The angular constraint function p_Θ ensures that (θ_f, ϕ_f) in some amino acid i and (θ_b, ϕ_b) in amino acid $(i+1)$ point toward one another. The potential is highest when they are antiparallel (that is, 180° apart), and decreases as the angle between them decreases. This idea is illustrated in Figure 3.6. *Again, note that each location $\vec{x_i}$ may have a different set of rotational parameters!*

3.3.1.2 Occupancy potentials

Occupancy potentials are in place to ensure that two residues do not occupy the same location in space. They are defined independently of orientation, and are merely a step function that constrains two (nonadjacent) Cα's to be at least K Å apart,

$$\psi_{occ}(\vec{u_i}, \vec{u_j}) = \begin{cases} 1 & ||x_i - x_j|| \geq K \text{ Å} \\ 0 & \text{otherwise} \end{cases} \tag{3.10}$$

Crystallographic symmetry [97] is managed in this occupancy potential. The occupancy potential is only non-zero if two amino acids – and *all symmetric copies* – are sufficiently far apart,

$$\psi_{occ}(\vec{u_i}, \vec{u_j}) = \begin{cases} 1 & \left(\min_{\substack{symmetric \\ transforms\ S}} ||x_i - S(x_j)|| \right) \geq K \text{ Å} \\ 0 & \text{otherwise} \end{cases} \tag{3.11}$$

In all experiments, we use a distance cutoff of $K = 3$Å (based on analysis of solved structures).

Finally, multiple chains in the asymmetric unit are also handled by ACMI-BP: separate chains are fully connected by edges enforcing occupancy constraints.

3.3.2 ACMI-BP's inference algorithm

The ultimate goal of ACMI-BP is producing a backbone trace: finding the labels $\mathbf{U}^* = \{u_i^*\}$ that maximize the probability of the local observational potentials and the global conformational potentials:

$$\mathbf{U}^* = \arg\max_{\mathbf{U}} \prod_{i \in \mathcal{V}} \psi_i(\vec{u_i}, \mathbf{M}) \times \prod_{(i,j) \in \mathcal{E}} \psi_{ij}(\vec{u_i}, \vec{u_j}) \tag{3.12}$$

However, as described in the previous chapter solving this exactly for large graphs with loops is intractable. ACMI-BP uses belief propagation (BP) to compute an approximation to the marginal probability $P_i(\vec{u_i}|\mathbf{M})$ for each amino acid i, then chooses the maximum marginal label for each residue as the final trace.

Recall that belief propagation is an inference algorithm that computes marginal probabilities using a series of local messages. At each iteration, a node (i.e., amino acid) computes an estimate of its marginal distribution (i.e., an estimate of the amino acid's location in the unit cell) as the product of its *local probability* and *all incoming messages*. This amino acid then passes a convolution of this product with the corresponding edge potential along each outgoing edge. Rewriting Equation 2.8 in terms of my protein model (EDM refers to the electron density map, i.e., I integrate over the entire map):

$$m_{i \to j}^{n}(\vec{u_j}) = \int_{\text{EDM}} \psi_{ij}(\vec{u_i}, \vec{u_j}) \times \frac{\hat{p_i^n}(\vec{u_i})}{m_{j \to i}^{n-1}(\vec{u_i})} \, d\vec{u_i} \tag{3.13}$$

Here, $\hat{p_i^n}(\vec{u_i})$ denotes the estimate of i's marginal (or i's *belief*) at iteration n, that is:

$$\hat{p_i^n}(\vec{u_i}) = \psi_i(\vec{u_i}, \mathbf{M}) \times \prod_{k \in \Gamma(i)} m_{k \to i}^{n}(\vec{u_i}) \tag{3.14}$$

Figure 3.7 illustrates the message-passing with a simple two-dimensional example. In this example, two residues' prior probabilities have their probability mass split among several peaks. Structural knowledge tells us that residue i must be next to residue j. In the first iteration, residue i passes a message to residue j, that indicates where residue i expects to find residue j (essentially, in a ring around residue i's peaks).

Messages in BP are probability distributions marginalized to the message recipient's random variables; that is, this message from residue i to residue j is a function over residue j's position in the density map. Residue j passes a message back to residue i indicating where j expects to find i. This example shows that in just two iterations, BP is able to reduce the number of peaks through the use of structural priors.

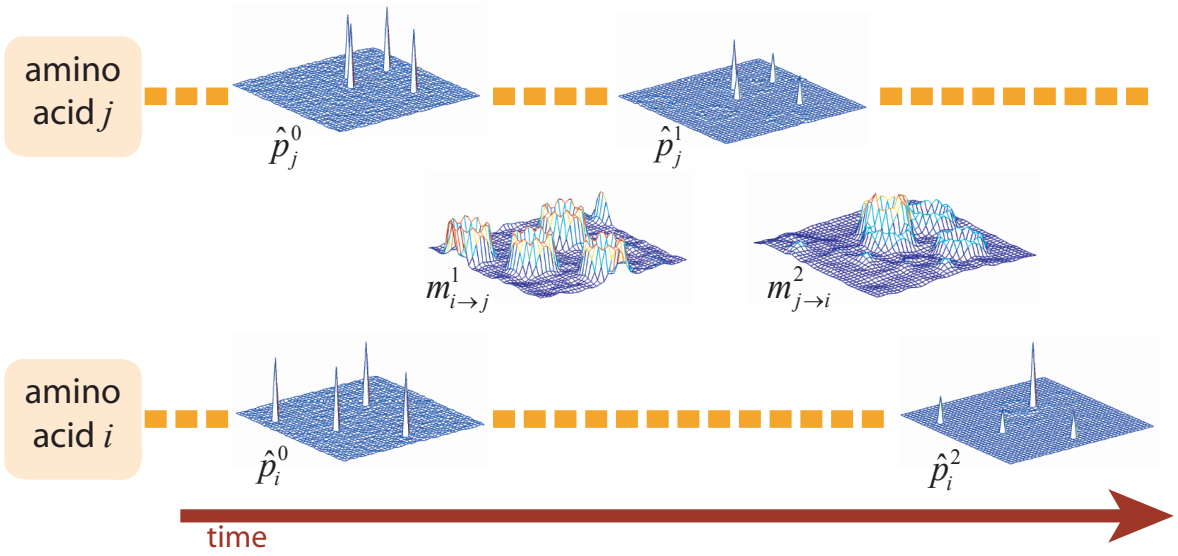


Figure 3.7 A simple example of message passing using belief propagation. Given prior probabilities $\hat{p_i^0}$ and $\hat{p_j^0}$, at each iteration, node i passes a message to a node j indicating i's belief of j's position. For example, a residue knows that an adjacent residue must be 3.8Å away; residue i's message to j consists of these 3.8Å "rings" of probability around i's peaks. As BP iterates, the matches that are structurally supported by other residues begin to emerge.

3.3.3 Technical challenges

Even with the computational savings afforded by BP compared to exact inference, the size and complexity of both the graph and the space of labels presented ACMI-BP with a number of implementation challenges. This section will briefly discuss some of these scaling issues. They will be described fully in Chapter 7.

3.3.3.1 Representation of potentials

The label associated with each residue is a continuously-valued, 7-dimensional variable. Nonparametric belief propagation (NBP) [106] is a variant of BP that can handle continuous-valued labels; previous work represented the belief as the sum-of-Gaussians. My work introduces Fourier-Series NBP, a variant of NBP which represents messages and belief as a set of 3D Fourier coefficients in Cartesian space, which offer a number of benefits for this problem domain. These benefits include natural treatment of periodic boundary conditions and symmetry, no explicit initialization required (as is required with the sum-of-Gaussians), and an efficient message-passing implementation.

3.3.3.2 Efficient message passing

Each message passed requires integrating over the entire unit cell, which naïvely takes running time of the order $\mathrm{O}(K^2)$, where K is the number of Fourier coefficients. Unfortunately, for a typical protein, K may be 10^6 to 10^7! For adjacency messages, it is not too much of a problem, as I only need to integrate over a thin spherical shell where ψ_{adj} is nonzero. However, for occupancy messages, this message computation time is significant. Fortunately, because the occupancy potential is only a function of the *distance* between the two connected residues, I can pass the message in $\mathrm{O}(K \log K)$ as a multiplication in Fourier-space.

That is, I can write the occupancy potential $\psi_{occ}(\vec{u_i}, \vec{u_j})$ as a function of the difference between the two amino acids in Cartesian space, i.e., $\psi_{occ}(x_i - x_j)$. Then, to compute an occupancy message from amino acid i, I take the Fourier transform of i's belief $\hat{p}_i(x_i)$ (marginalized over rotations to be a function of location x_i), and multiply it by the (precomputed) Fourier transform of the occupancy potential (the step function). That is:

$$m_{i \to j}(\vec{x_j}) = \mathcal{F}\left[\hat{p}_i(x_i)\right] \times \mathcal{F}\left[\psi_{occ}(x_i - x_j)\right] \tag{3.15}$$

Notice that these occupancy messages are *uniform* over rotations.

3.3.3.3 Occupancy message aggregation

Because ACMI-BP's protein graph is fully connected, in each iteration $\mathrm{O}(N^2)$ messages need to be computed and stored, where N is the number of amino-acid residues in the protein. As each message is a probability distribution over the entire density map, this is demanding computationally and in terms of storage. However, the outgoing structural messages (see Equation 3.13) at a given node are all quite similar: they only differ in the denominator, which serves to avoid double-counting, making the method exact in tree-structured graphs [118].

In loopy graphs, this double-counting is unavoidable. I can save a significant amount of work if I aggregate all the non-bonded residues, sending them instead a single structural message. This aggregate message simply drops the denominator for each outgoing occupancy message. This approximation, coupled with an accumulator (see Figure 3.8) that stores the product of all nodes outgoing occupancy messages. This approximation reduces ACMI-BP's runtime and memory use to $\mathrm{O}(N)$ per iteration. Complete descriptions and additional experiments using message aggregation are given in Chapter 7.

Combined, these three BP optimizations allow ACMI-BP to handle large proteins with large unit cells. Typical run times (for BP inference) vary from several hours to several days.

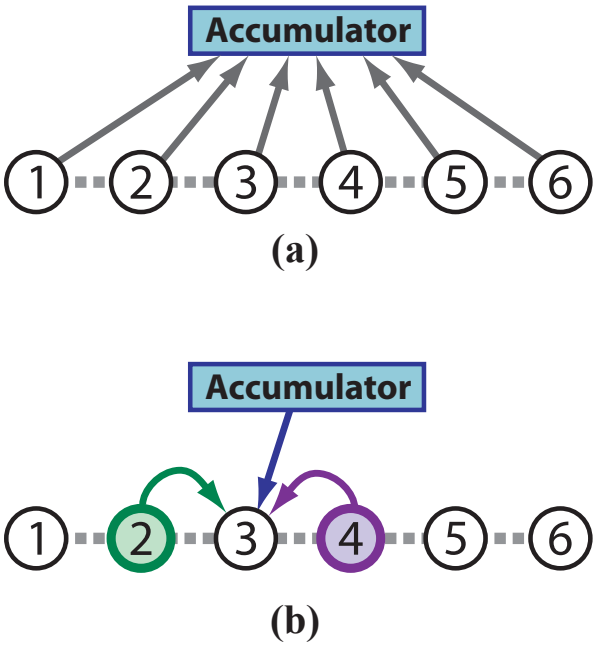


Figure 3.8 An illustration of occupancy message aggregation: (a) Every amino-acid in the protein sends an *approximate* occupancy message to an accumulator. (b) The accumulator allows updating a node's (in this example, node 3) belief in constant time, as the accumulator product times incoming adjacency messages (from nodes 2 and 4), divided by occupancy messages it *should not* have received, but did from the accumulator product (here, from nodes 2, 3, and 4).

3.4 Experiments

I obtained a set of ten model-phased[2] electron-density maps from the Center for Eukaryotic Structural Genomics at the University of Wisconsin–Madison. Details of the dataset – including PDB accession codes – are shown in Section A.2, in Appendix A. I removed maps in the testset from my database of PDB structures before testing. To test ACMI's performance on poor-quality (2.5+ Å) data, I downsampled these maps to $R_0 = 2.5$, 3, 3.5, and 4Å resolution by *smoothly* diminishing the intensities of higher-resolution reflections. Details of the truncation are given in Section A.2.

I compare the performance of ACMI (that is, the sequential application of ACMI-FF and ACMI-BP) on these maps to two other automated techniques specialized to low-resolution maps: Ioerger's TEXTAL [55], and Terwilliger's RESOLVE [110, 111]. I measure each algorithm's prediction using

[2]A model-phased map uses the phasing from the final deposited model as opposed to the experimentally estimated phases to construct the density map. Model-phased maps have no phasing artifacts, and are typically of better quality than the experimentally phased density map.

three different metrics: (a) Cα RMS error between predicted model and ground truth (the crystallographer's solution), (b) the fraction of Cα's in the model that were located, and (c) the fraction of Cα's that were located *and* correctly identified.

The distinction between the second and third metric is that often, RESOLVE (and TEXTAL to a lesser extent) will identify some portion of the backbone, but will be unable to align it to the provided sequence. Amino-acids in this portion would be identified as correct using metric (b), but incorrect under metric (c). ACMI, too, will occasionally misidentify residues (for example, swapping two helices).

The results at each resolution are summarized in Figure 3.9. TEXTAL was unable to run on one protein's density maps (at any resolution) – rather than including a terrible score for this map, I gave the benefit of the doubt to TEXTAL and only report results on the nine maps on which it ran. In terms of RMS error (Figure 3.9a), my algorithm consistently outperforms TEXTAL at all resolutions tested. Using a two-tailed paired t test, ACMI outperforms TEXTAL with p values of 0.091, 0.057, 0.012 and 0.11 at 2.5, 3, 3.5, and 4Å, respectively. RESOLVE performs on par with ACMI at 2.5Å resolution; however, at 3, 3.5 and 4Å, ACMI's performance is much better: a two-tailed t test yields p values of 0.0068, 0.00002 and 0.00004, respectively (both of these t tests only take into account RMS error and not chain coverage).

Figure 3.9b shows that the percent of the chain covered was roughly equivalent for the three approaches. However, Figure 3.9c shows that my approach is much better than the others at identifying the proper residue type at a particular location. It is important to point out that these related methods are not optimizing residue-identification accuracy. RESOLVE, for example, will often return a long chain of alanine residues if it cannot identify sidechains, but still gives the correct backbone structure overall. This illustrates a significant difference between ACMI and these alternate approaches: TEXTAL and RESOLVE build a backbone model, then attempt to align the protein sequence to it. ACMI, alternatively, uses the sequence of the protein to construct the model. The result is better identification of amino acids in the map.

Figure 3.10 shows scatterplots in which each individually solved electron-density map is a point. The x-axis indicates ACMI's error; the y-axis indicates TEXTAL's (or RESOLVE's) error. All points above the diagonal line correspond to maps where ACMI outperformed TEXTAL (or RESOLVE). On the majority of structures, ACMI's interpretation has a lower RMS error then both of the other algorithms. ACMI is outperformed by RESOLVE on some high-resolution maps, however, ACMI currently does not perform any post-processing on predicted backbones (e.g. real-space refinement [86], energy minimization [9]); also, Cα's are restricted on a grid, limiting accuracy to the grid spacing.

One advantage of ACMI's probabilistic framework is that, in addition to returning a putative trace, ACMI also returns a confidence (i.e., probability) level of each predicted residue. This confidence informs the crystallographer what areas in the map need improvement; alternatively, a high confidence partial trace could be used to improve phasing. Figure 3.11 illustrates this in an example trace at 3.5Å resolution, on a structure consisting of two chains of 124 residues each. This is ACMI's sixth-best (of the ten) traces at this resolution: ACMI finds nine segments with a Cα RMS deviation of 2.3Å, covering 94% of the backbone. The model's shading indicates the likelihood of its prediction for each Cα location.

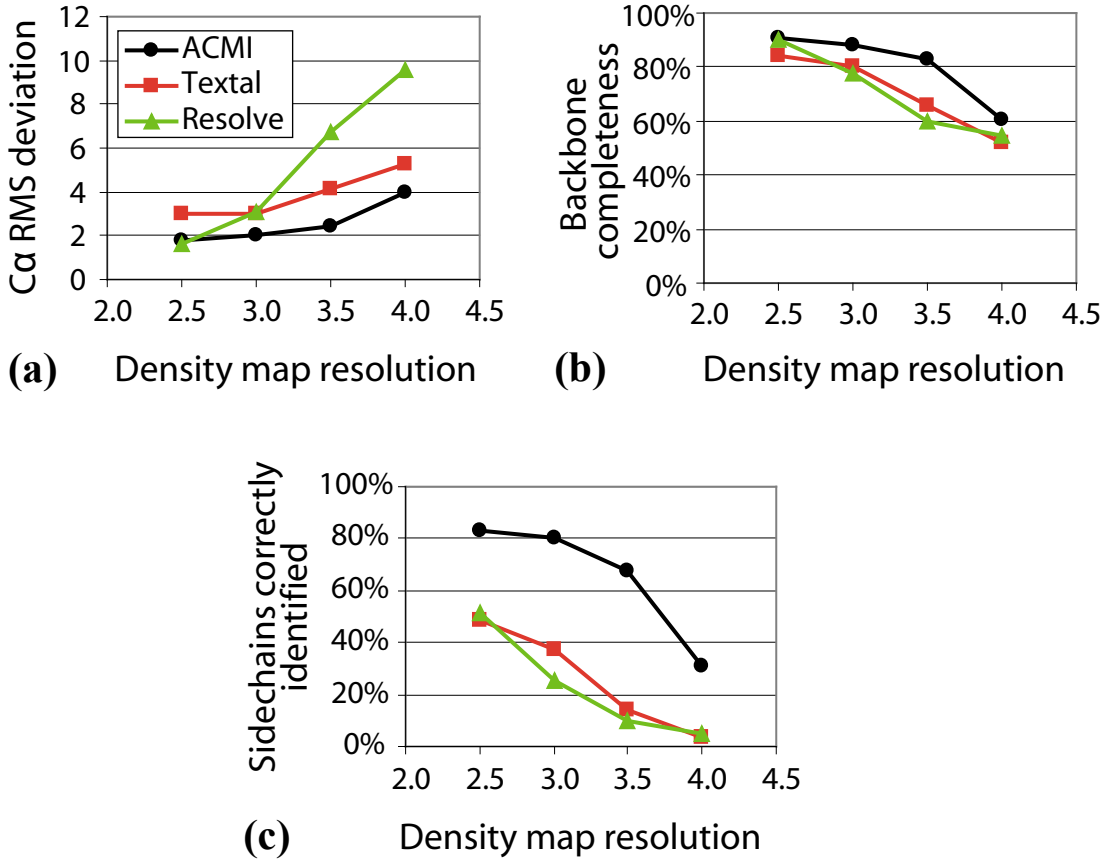


Figure 3.9 Graphs comparing ACMI's, TEXTAL's, and RESOLVE's interpretation in terms of (a) average RMS Error, (b) average percent of the chain located, and (c) average percent of residues correctly identified.

3.5 Conclusions and future work

I describe ACMI, a tool for automatically tracing protein backbones especially designed for poor-quality electron-density maps. ACMI combines a local matching procedure and a global constraint procedure in a probabilistic framework that can efficiently infer the locations of backbone atoms in an electron-density map. The algorithm provides accurate traces even in poor resolution electron-density maps, outperforming both TEXTAL and RESOLVE above 3Å map resolution.

One major shortcoming is the significant compute time required by ACMI-FF's local (5-mer) matching procedure. I need to search for approximately three 5-mer fragments per residue; for each fragment I consider ≈ 1900 rotations[3]. Even for medium-sized unit cells, this takes on the order of

[3]At 20° discretization, there are $18 \times 9 \times 18 = 2916$ 3D rotational samples to consider. Using a shortcut suggested by Mitchell [80], I am able to reduce this by around 40%.

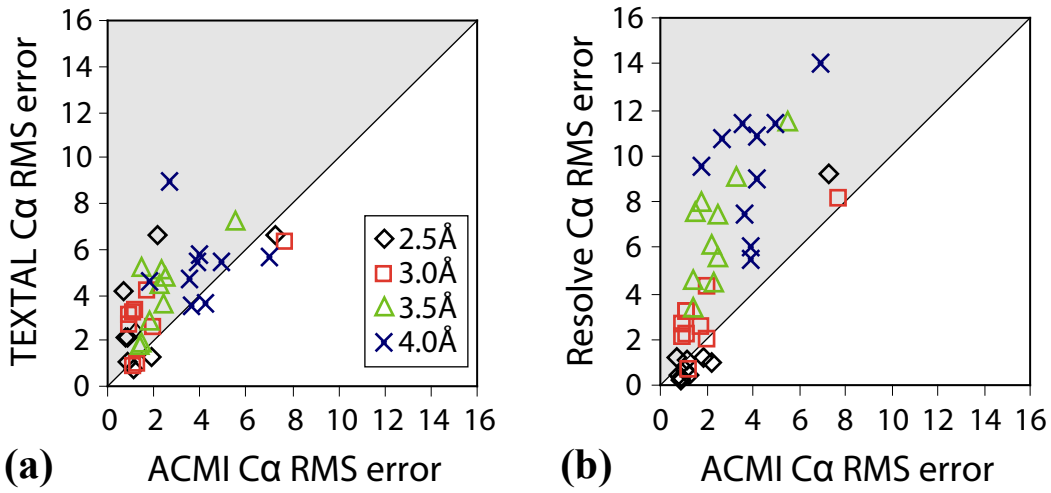


Figure 3.10 A scatterplot showing the performance on a protein-by-protein basis, of ACMI versus (a) TEXTAL and (b) RESOLVE. Each mark is an interpreted map; the shaded half of each scatterplot (above the diagonal) shows the region where ACMI provided a more-accurate backbone trace.

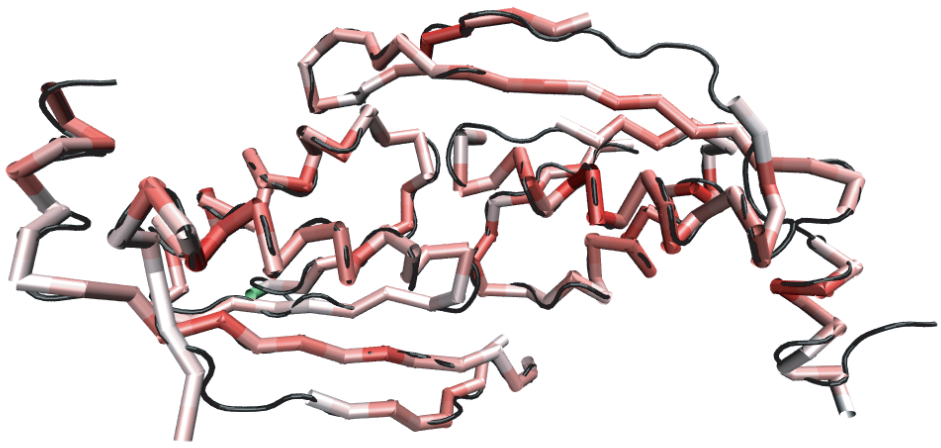


Figure 3.11 An illustration of predicted versus actual structure on ACMI's sixth-best prediction (out of ten) at 3.5Å resolution. The thin continuous coil is the actual structure, while the thicker segmented chain is ACMI's prediction. The predicted structure is colored by log-likelihood, where Cα placements with higher likelihood are shaded darker.

CPU-weeks; larger proteins take CPU-months. ACMI-FF exploits parallelism, running overnight, using the spare cycles from desktop computers [112]. However, I would like to investigate the use of machine learning algorithms, such as support vector machines or neural networks, to quickly match a 5-mer into the density map. I also would like to explore alternative feature representations.

Additionally, I would like to investigate improved methods for managing non-crystallographic symmetry. Many maps have this type of symmetry, that is, where the protein forms a multimeric complex in the asymmetric unit. Every copy must be found by the automated method or the crystallographer, although often all copies are in identical or nearly identical conformations. Currently, my global constraint model, ACMI-BP, places all chains simultaneously, but does not use the knowledge that chains are likely to take similar conformations. This knowledge, however, is used by crystallographers in solving such maps, and is valuable in solving very poor density maps. In the future, I would like to add to ACMI-BP the ability to infer the conformation of just a single protein chain and one or more transformations relating these multiple copies.

By providing accurate interpretations from lower-resolution maps, ACMI reduces the burden on crystallographers when only poor-quality density map data is available. Even when it may be possible to obtain high-resolution density-map data, ACMI allows significant cost savings by finding accurate solutions with poor-quality maps. In doing so, ACMI speeds up the process of high-throughput protein structure determination.

Chapter 4

Improved Template Matching Using Spherical Harmonics

The previous chapter described ACMI-FF and ACMI-BP, a two-step method for automatically producing a backbone trace from an electron-density map. A critical component in this method is ACMI-FF, the *local match* component. ACMI-FF independently searches the density map for a set of template fragments, using Fourier convolution to quickly search the entire map for some rotation of a template. This chapter describes a significant improvement to this template matching over my original approach to template matching (Section 3.2).

This chapter's approach to template matching, ACMI-SH, uses the spherical-harmonic decomposition of the template and some region of the density map. This enables rapidly searching over all *rotations* of some fragment at a single location in the density map. An initial filtering algorithm, unusable in the Fourier convolution framework, reduces computational time by eliminating a majority of points without performing a costly rotational search. Combining ACMI-SH's local search with ACMI-BP's inference enables more accurate interpretation of poor quality density-map data. Much of the material in this chapter was previously published [28].

4.1 The goal of template matching

Recall that ACMI-BP builds a probabilistic protein-backbone model, where the probability of some backbone model $\mathbf{U} = \{u_i\}$ (where u_i is the position and orientation of the ith Cα) is given as

$$P(\mathbf{U} = \{u_i\}) \propto \prod_{\text{amino-acid } i} \psi_i(u_i) \times \prod_{\substack{\text{amino-acids } i,j \\ i \neq j}} \psi_{ij}(u_i, u_j) \tag{4.1}$$

For clarity, dependence on the density map $\mathbf{M}$ has been dropped. The first product models how well an amino acid matches some location in the density map; the second models the global structural constraints on the protein.

The first term in this equation, the vertex potential ψ_i, results from independently searching the map for a set of templates corresponding to amino-acid i. It can be thought of as a "prior probability" on each alpha carbon's location, given the density map. Another way to think of this is as there being an "amino-acid-i detector" for each amino-acid i in the protein.

The second term in this equation, the edge potential ψ_{ij}, enforces structural constraints on the protein. An inference algorithm (introduced in Chapter 3 and described in detail in Chapter 7)

finds the most likely location of each C_α, given the density map. Specifically, for each amino acid i in the protein, ACMI's inference algorithm returns a probability distribution over Cα i's location.

This chapter is concerned with the computation of the first term in Equation 4.1's product, the vertex potentials ψ_i. Accurate "amino-acid detectors" are important to producing an accurate backbone model. These amino-acid detectors search the density map for a set of templates corresponding to each amino acid. To construct these templates, I consider a 5-mer (that is, a 5-amino-acid sequence) centered at each position in the protein sequence, and build a set of template pentapeptides (that is, 5-amino-acid *structures*).

The previous chapter's ACMI-FF clusters these pentapeptides, searching the map for a single representative from each cluster. To quickly match these templates to the density map, ACMI-FF uses Fourier convolution (like that of Cowtan's FFFEAR [17]) to compute the *squared density difference* of one rotation of a template to the entire density map. I then use a tuning set to convert squared density differences into a probability distribution over the electron-density map. Although efficient, one disadvantage of ACMI-FF is that I am forced to search the entire density map for some template. The Fourier convolution does not allow us to search in only some small region of the map.

The remainder of this chapter describes ACMI-SH, an alternate approach to searching the density map for occurrences of some template. Using spherical-harmonic decomposition, ACMI-SH rapidly searches all rotations of a template at a single location. This framework lets me limit the locations in which I search to some small subset of the entire density map, addressing a primary shortcoming of the previous chapter's approach. In addition, ACMI-SH's improved efficiency lets me search for more templates and at higher angular resolution than the previous chapters template-search method.

4.2 Spherical harmonics and the fast rotation function

My new method for template-matching is based on spherical-harmonic decomposition and is similar to the fast rotation function used in molecular replacement [20, 113], as well as for shape matching in other domains [41, 49].

Spherical harmonics $Y_l^m(\theta, \phi)$, with order $l = 0, 1, \ldots$ and degree $m = -l, -(l-1), \ldots, l$, are the solution to Laplace's equation[1] in spherical coordinates. They are analogous to a Fourier transform, but on the surface of sphere. They form an orthogonal basis set on the sphere's surface. Any spherical function $f(\theta, \phi)$ can be written

$$f(\theta, \phi) = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} a_{lm} \cdot Y_l^m(\theta, \phi) \tag{4.2}$$

Figure 4.1 illustrates the real and imaginary components of some low-order spherical harmonics. Figure 4.2 shows an example of how a spherical function is decomposed using harmonic coefficients.

[1] Laplace's equation, $\bigtriangledown^2 \phi = 0$ is a fundamental partial differential equation [30], arising in many different scientific fields, including fluid dynamics, electromagnetics, and gravitation.

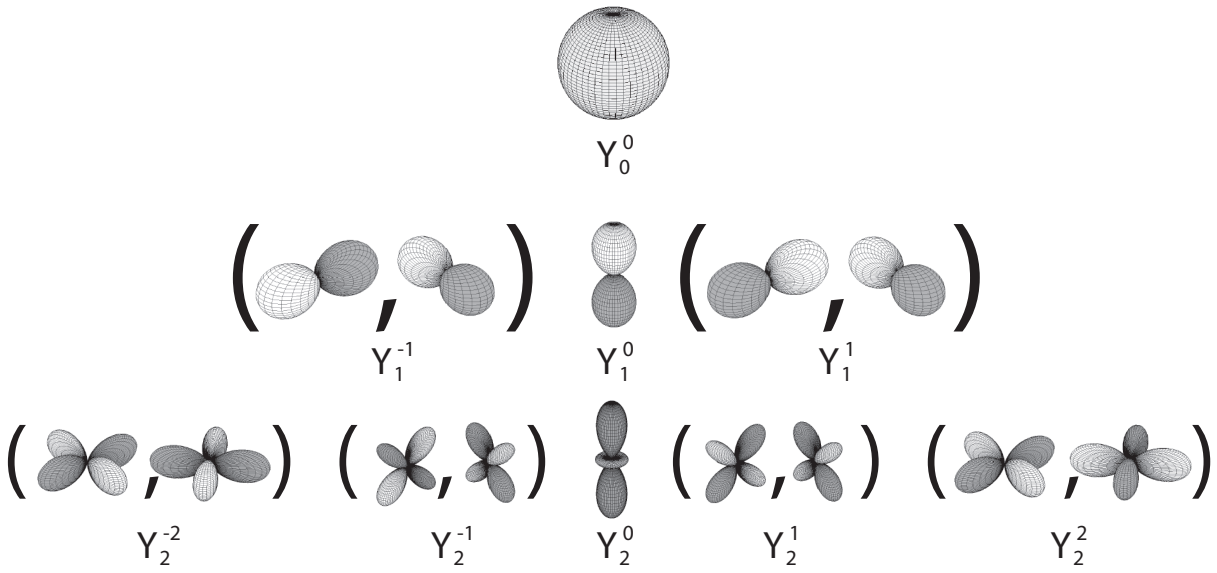


Figure 4.1 The real and imaginary components of several low-order spherical harmonics. These basis functions are illustrated such that the radius at a particular angular coordinate indicates the value at that particular location. Colors indicate sign (gray positive and white negative).

One important property is that the *rotation* of spherical harmonics of some order is simply a linear combination of spherical harmonics of the same order. So, using the notation $\mathbf{R}(\vec{r}) \cdot f(\theta, \phi)$ to denote the rotation of a function f by the angles $\vec{r} = \langle \alpha, \beta, \gamma \rangle$,

$$\mathbf{R}(\vec{r}) \cdot Y_l^m(\theta, \phi) = \sum_{k=-l}^{l} Y_l^k(\theta, \phi) \cdot D_{km}^l(\vec{r}) \tag{4.3}$$

These linear coefficients $D_{km}^l(\vec{r})$ refer an entry into the Wigner D-matrix (see [119]). This matrix is rapidly computed using separation of variables and an FFT-like recurrence.

A key advantage of this decomposition arises when I want to rotationally align two spherically defined functions [69]. That is, given (real) functions $f(\theta, \phi)$ and $g(\theta, \phi)$ on the sphere, I want to compute the *cross correlation* between them as a function of rotation angles $\vec{r}$,

$$C_{fg}(\vec{r}) = \int_0^{2\pi} \int_0^{\pi} f(\theta, \phi) \cdot \mathbf{R}(\vec{r}) \cdot g(\theta, \phi) \cdot sin\theta \, d\theta \, d\phi \tag{4.4}$$

If the functions f and g are band-limited to some maximum bandwidth B (or can be reasonably approximated as such), then expanding these functions in terms of their spherical-harmonic decomposition gives this cross-correlation in terms of the Wigner D-matrix (a and b refer to the

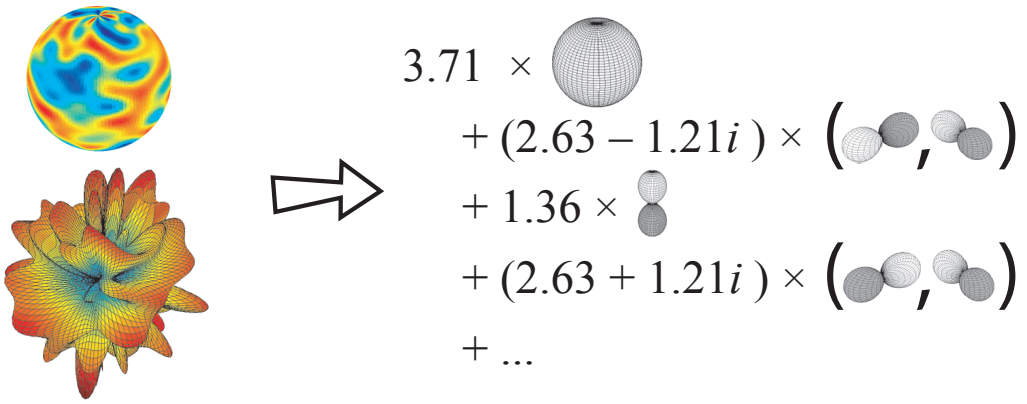


Figure 4.2 An overview of spherical-harmonic decomposition. Given the spherically defined function on the left – illustrated using both shading (top) and radius (bottom) to illustrate function values – I compute its decomposition as a set of spherical-harmonic coefficients.

spherical-harmonic coefficients of f and g, respectively),

$$C_{fg}(\vec{r}) = \sum_{l=0}^{B-1} \sum_{m=-l}^{l} \sum_{n=-l}^{l} a_{l(-m)} \cdot b_{l(-n)} \cdot (-1)^{m-n} D^{l}_{mn}(\vec{r}) \tag{4.5}$$

Thus, computing $C_{fg}(\vec{r})$ is simply a matter of combining the spherical-harmonic coefficients of f and g, and computing the *inverse* of the transformation used to compute the Wigner D's above. A full derivation is shown by Kostelec and Rockmore [69].

Several different "fast rotation" algorithms exist to quickly compute this cross correlation $C_{fg}(\vec{r})$. If functions f and g have maximum bandwidth B, then these fast rotation functions compute this cross correlation given the spherical-harmonic decomposition of f and g in $\mathrm{O}(B^4)$ time or $\mathrm{O}(B^3 \log B)$ time as opposed to the naïve $\mathrm{O}(B^6)$.

4.3 A method for fast template matching

I derive an improved vertex potential from this fast rotation function. An overview of my local-match procedure appears in Algorithm 4.1, and is illustrated in Figure 4.3.

4.3.1 Overview of the approach

When searching for some pentapeptide, I begin by computing the density I would expect to see given the pentapeptide (one models each atom with a Gaussian sphere of density). I then interpolate this calculated density in concentric spherical shells (uniformly gridding $\theta - \phi$ space) extending out to D Å (typically 5 Å, chosen to cover most of the density in an average pentapeptide) in 1Å steps.

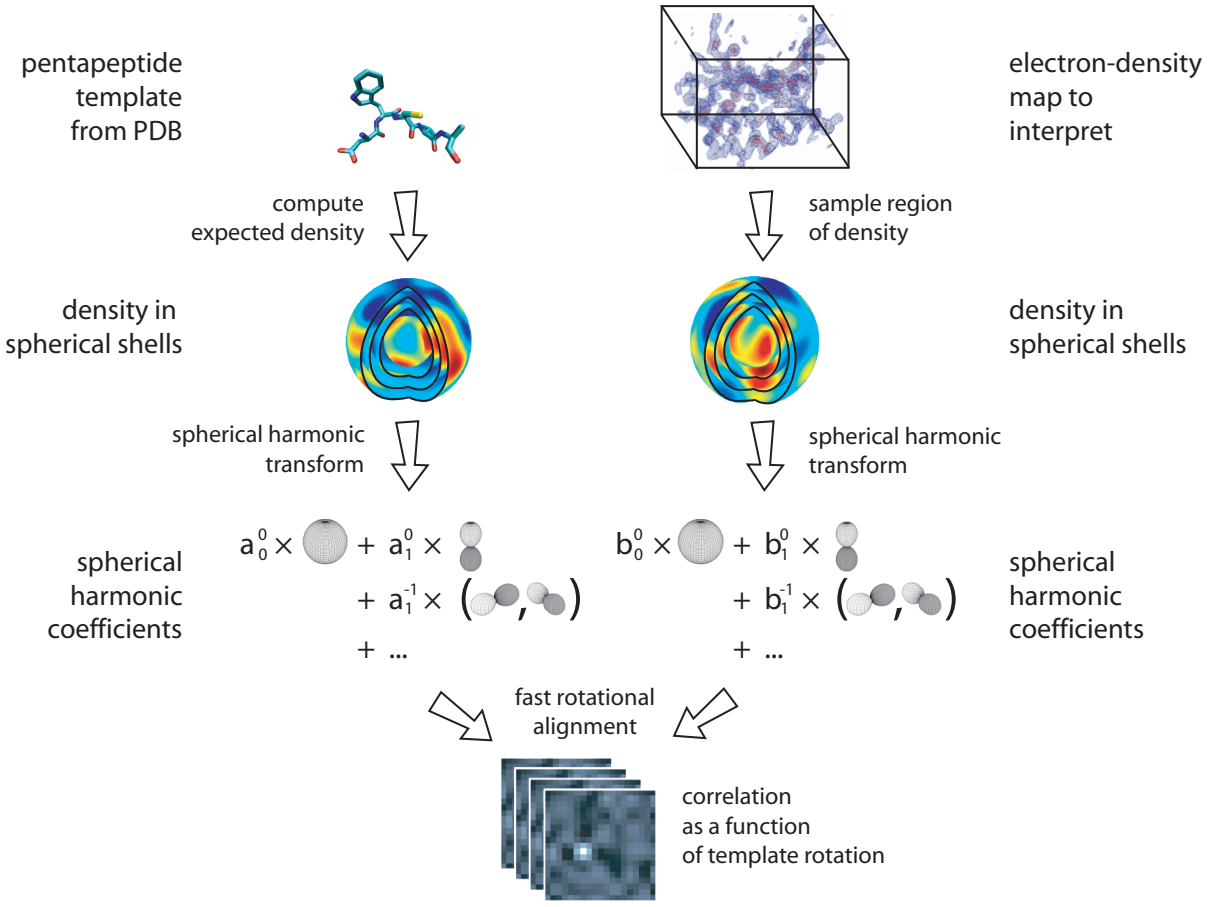


Figure 4.3 ACMI-SH's improved template-matching algorithm. Given some pentapeptide template (left) the expected electron-density is calculated. In the map (right), a spherical region is sampled. Spherical harmonic coefficients are calculated for both, and the fast rotation function computes cross correlation as a function of template rotation.

A fast spherical-harmonic transform computes spherical-harmonic coefficients corresponding to each spherical shell using a recursion similar to that used in fast Fourier transforms [43].

Similarly, I interpolate the density map using the same set of concentric spherical shells around some grid point, and again, take the spherical-harmonic transform of each spherical shell's density. Given these two sets of spherical-harmonic coefficients – one corresponding to the template and one corresponding to some location in the density map – the previous section's fast rotation alignment computes the *cross correlation* over all rotations of the template pentapeptide, for each spherical shell. ACMI-SH's fast rotational correlation uses the implementation of Rockmore and Kostelec [69].

Algorithm 4.1 ACMI-SH's template matching.

input: amino-acid sequence Seq, density map $\mathbf{M}$
output: vertex potentials $\psi_i(\vec{y}, \vec{r})$ for $i = 1 \ldots N$
// ψ_i is a function of location $\vec{y} = (x, y, z)$ and rotation $\vec{r} = (\alpha, \beta, \gamma)$

// Learn parameters to convert correlations to probabilities
$(\mu_{CC}, \sigma_{CC}) \leftarrow$ learn-from-tuneset()

foreach *residue* i **do**
  // Search for pentapeptides corresponding to amino-acid i
  $\mathbf{PDBfrags}_i \leftarrow$ lookup-in-PDB$(Seq_{i-2:i+2})$

  // For each corresponding pentapeptide template
  foreach $frag \in \mathbf{PDBfrags}_i$ **do**
    // Compute the template's density and its decomposition
    $template \leftarrow$ compute-dens$(frag)$
    $templCoef \leftarrow$ SH-transform$(template)$

    // Now scan over the density map
    foreach *point* $\vec{y_j} \in \mathbf{M}$ **do**
      // if $\vec{y_j}$ doesn't pass filtering criteria, don't search
      if is-filtered-out$(\vec{y_j})$ **then next** $\vec{y_j}$

      // Sample the density around $\vec{y_j}$; take its decomposition
      $signal \leftarrow$ sample-dens-around$(\vec{y_j})$
      $sigCoef \leftarrow$ SH-transform$(signal)$

      // Align density regions using the fast rotation function
      $\mathbf{CC} \leftarrow$fast-rotate$(templCoef, sigCoef)$
      $CC^\star \leftarrow$sum-over-spherical-shells$(\mathbf{CC})$

      // For each rotation of the template
      foreach *rotation* $\vec{r_k} \in \mathbf{R}$ **do**
        // Convert correlation at rotation r_k to a probability
        $z_k \leftarrow (\mu_{CC} - CC^\star_k)/\sigma_{CC}$
        $p_{null} \leftarrow$ normCDF (z_k)
        $\psi_i(\vec{y_j}, \vec{r_k}) \leftarrow (1 - p_{null})/p_{null}$
      end
    end
  end
end

This gives a cross-correlation (as a function of rotation) for each spherical shell. To combine scores over the entire region, I take, at each rotation $\vec{r}$, the weighted sum of each shell's correlation coefficient. The weight of each shell corresponds to the surface area of that shell, thus, the outmost shells have the greatest weight. That is, given the correlation of each spherical shell, $\mathbf{CC}(\vec{r}) = \{CC^1(\vec{r}), \ldots, CC^D(\vec{r})\}$, I compute the correlation over the entire region as:

$$CC^\star(\vec{r}) = \frac{1}{\sum_{d=1}^{D} 4\pi d^2} \sum_{d=1}^{D} 4\pi d^2 \times CC^d(\vec{r}) \tag{4.6}$$

This gives a single correlation coefficient for each rotation of the fragment, over all spherical shells.

After computing the cross correlation, I compute the vertex potential ψ_i as the probability that a particular cross-correlation value was *not* generated by chance. That is, I assume that the distribution of the cross correlation between some template's density and some random location in the density map is normally distributed with mean μ and variance σ^2,

$$C_{fg} \sim \mathcal{N}(x; \mu, \sigma^2) \tag{4.7}$$

I estimate these parameters μ and σ^2 by computing cross correlations between the template and random locations in the map. Given some cross correlation x_c, I compute the expected probability that I would see score c_i or higher by random chance,

$$p_{null}(x_c) = P(X \geq x_c; \mu, \sigma^2) = 1 - \Phi((x_c - \mu)/\sigma) \tag{4.8}$$

Here, $\Phi(x)$ is the normal cumulative distribution function. Each amino-acid's potential is then $(1 - p_{null})/p_{null}$.

For a given template, ACMI-SH scans the density map $\mathbf{M}$, centering the template at every location $(x_i, y_i, z_i) \in \mathbf{M}$. At each location, I sample concentric spheres of density around (x_i, y_i, z_i), take the spherical-harmonic transform, and compute the cross correlation between the template and density map around (x_i, y_i, z_i) as a function of 3D rotation angles $\vec{r} = (\alpha, \beta, \gamma)$.

ACMI-BP's inference algorithm only stores a single probability value and a single rotation per location, assuming a Gaussian distribution in rotational space about this stored location. So for the final step in my template matching, at each location (x_i, y_i, z_i), I store the maximum probability over all rotations, as well as the rotation corresponding to this probability. However, this is a limitation of the inference algorithm, not of template matching. ACMI-SH's template matching gives a probability distribution over *all* locations and rotations of the template fragment.

4.3.2 Advantages of rotational "convolution"

In ACMI-FF, searching for some template in an $X \times X \times X$ electron-density map (that is, with X^3 grid points for which density values are provided) at an angular bandwidth B requires that I search the density map for $O(B^3)$ rotations of each fragment (the angular bandwidth B here corresponds to an angular "grid size" of $180^\circ/B$ in Chapter 3's ACMI-FF). Searching for a single rotation over the map is done using 3 FFT's, which run in $O(X^3 \log X)$. Thus, searching the density map using the previous chapter's template search takes running time $O(X^3 \log X B^3)$.

Using spherical-harmonic decomposition, ACMI-SH requires that I search for the fragment in $O(X^3)$ locations. Searching for all rotations of some template at one location takes $O(B^3 \log B)$, giving ACMI-SH a running time of $O(X^3 B^3 \log B)$.

In this application, though, one can think of B as "fixed" with respect to map size, since B only depends on the size of the template pentapeptide. As protein size (and consequently density map size) increases, ACMI-SH shows improved scaling, $O(X^3)$ versus $O(X^3 \log X)$.

"Convoluting[2]" in rotational space rather than Cartesian space (as in FFFEAR [17]) offers several advantages. First I only have to search the *asymmetric unit* of the protein crystal – that is, only the smallest non-repeated portion of the density map – rather than the entire map. This alone may offer six to eight-fold runtime savings.

Additionally, ACMI-SH allows the use of a "first-pass filter" that only considers some small portion of the density map over which I perform a rotational search. Because ACMI-FF searches the entire map at once using a Fourier convolution, it requires that I search over the entire density map for each fragment, even if I know a template lies in some small area. Using ACMI-SH, I instead search all of rotational space at once, allowing me to limit the (x, y, z) locations at which I search. A comparison of several such filters is presented in the next section.

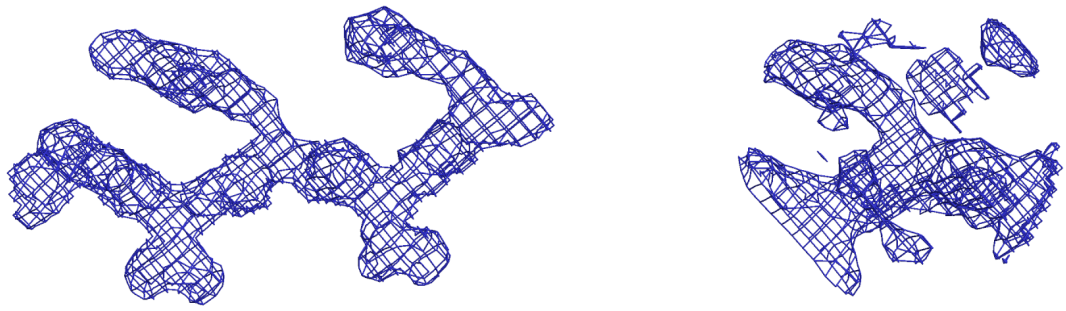
4.3.3 Modified pentapeptide templates

There are other changes between ACMI-FF and ACMI-SH as well. Because ACMI-SH samples spherical density shells, the template for which I am searching is a fixed-size sphere around the center of each template structure. This sphere includes many (but not all) atoms from the pentapeptide; in addition, it includes atoms from other portions of the protein located nearby. This contrasts with ACMI-FF, where each template was arbitrarily shaped: a mask was extended to 2.5Å away from each atom in the template pentapeptide.

Figure 4.4 compares one such fragment: Panel A shows ACMI-FF's template density for some pentapeptide, while Panel B shows ACMI-SH's template density for the same pentapeptide. I feel ACMI-SH's representation is advantageous as it captures the *context* of each pentapeptide: for example, if some 5-mer *always* occurs on the surface of a protein, all of that 5-mer's templates will be on the protein surface, and will be reflected in the cross-correlation scores. That is, a template on the surface of a protein will match best to regions of the map on the surface of the protein. Alternatively, one could use a fixed-size sphere to align a template to the map, then compute the correlation coefficient over some arbitrary-shaped region; in my experience this produces no improvement in matching accuracy, and incurs non-trivial overhead.

A final difference between ACMI-FF and ACMI-SH is that – in ACMI-FF – I cluster the template structures (from the PDB) to produce a *minimal subset* for which I search. ACMI-SH no longer clusters these templates. In ACMI-FF, clustering serves mainly to reduce computational costs. Due to improved efficiency of ACMI-SH, I am able to search for a greater number of fragments than before. Even if I wanted ACMI-SH to cluster templates, I would run into trouble.

[2]I use this term loosely to refer to computation of both ACMI-FF's squared-density-difference (which is computed via convolution) and ACMI-SH's cross-correlation.



(a) **(b)**

Figure 4.4 Differences between ACMI-FF's and ACMI-SH's density templates. (a) ACMI-FF's templates include all density within 2.5Å of any template atom, while (b) ACMI-SH's templates include all density within 5Å of the center Cα, which may include density from atoms not in the pentapeptide.

ACMI-FF clusters pentapeptides using RMS deviation as a distance metric. In ACMI-SH, templates are now fixed-sized spheres, which often includes atoms not in the pentapeptide. This makes RMS deviation, which does not take these atoms into account, an ineffective measure for similarity between two templates. A measure which took these atoms into account – such as cross-correlation between templates or searching for atom correspondences – would incur significant performance penalties (hierarchical clustering of N objects requires $O(N^2)$ comparisons; in ACMI-SH often $N > 100$).

Therefore, ACMI-SH simply searches for *every* template pentapeptide in my non-redundant protein data bank subset corresponding to a particular 5-mer sequence.

4.4 Results

This section evaluates ACMI-SH using four different performance measures. The first two measures are simple tests of ACMI-SH: I first show the error introduced by band-limiting density templates, then I compare several different first-pass filters. My third test compares ACMI-FF to ACMI-SH, in terms of matching accuracy and running time as rotational sampling (the resolution of the θ–ϕ grid) varies. Finally, I use both ACMI-FF and ACMI-SH's matching routines as input to ACMI-BP's inference engine, and compare the resulting protein models to other approaches. The experimental setup in this final section is the same as used in the previous chapter.

As in the previous chapter, I use a set of ten model-phased electron-density maps from the Center for Eukaryotic Structural Genomics at the University of Wisconsin–Madison. Details of the dataset – including PDB accession codes – are shown in Section A.2, in Appendix A. Maps in my testset were removed from my non-redundant PDB subset before testing. To test ACMI's

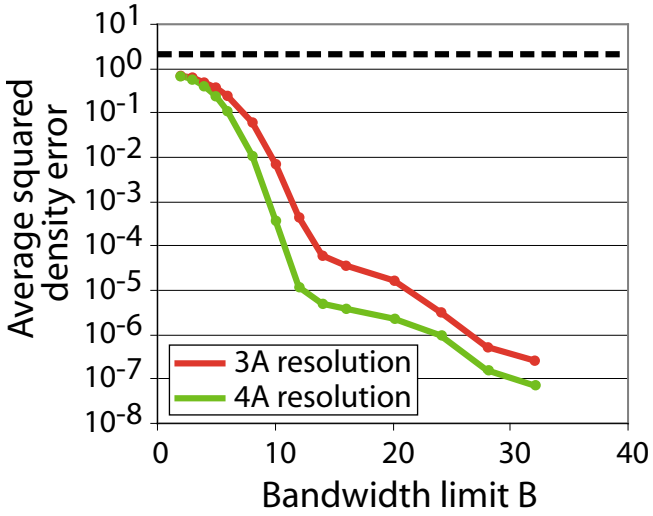


Figure 4.5 The average squared density difference between a region of sampled density and the bandwidth-limited region. The dotted line shows the error between two random regions.

performance on poor-quality (2.5+ Å) data, I downsampled these maps to 3 and 4Å resolution by *smoothly* diminishing the intensities of higher-resolution reflections. Details of the truncation are given in Section A.2.

4.4.1 Errors in band-limiting density

Rotationally aligning two regions of density using spherical harmonics requires that I compute spherical harmonics of both the density map and the template density up to some band limit B. This band-limited signal will be somewhat different than the original signal. Figure 4.5 shows the average *squared density difference* between the original sampled density and the bandwidth-limited density as B is varied. The dotted line in this figure shows the squared density between two random regions, as a baseline (this measure does not depend on bandwidth limit or resolution).

This figure shows a bandwidth limit $B = 12$ accurately models the original density, with density difference $< 10^{-3}$ for 3Å resolution maps and $< 10^{-4}$ for 4Å resolution maps. The difference between two random signals is around 2.

As further support, Trepani and Navaza provide a rule of thumb for the bandwidth limit in Patterson maps [113] (an experimental map related to the electron-density map), where the band limit B relates to the density-map resolution d and the radius r by the formula

$$B \approx 2\pi r/d \tag{4.9}$$

In this application, where I use a radius of 5Å (thus $B \approx 10$ for a 3Å map and 8 for a 4Å map), the rule seems to closely match my result in Figure 4.5.

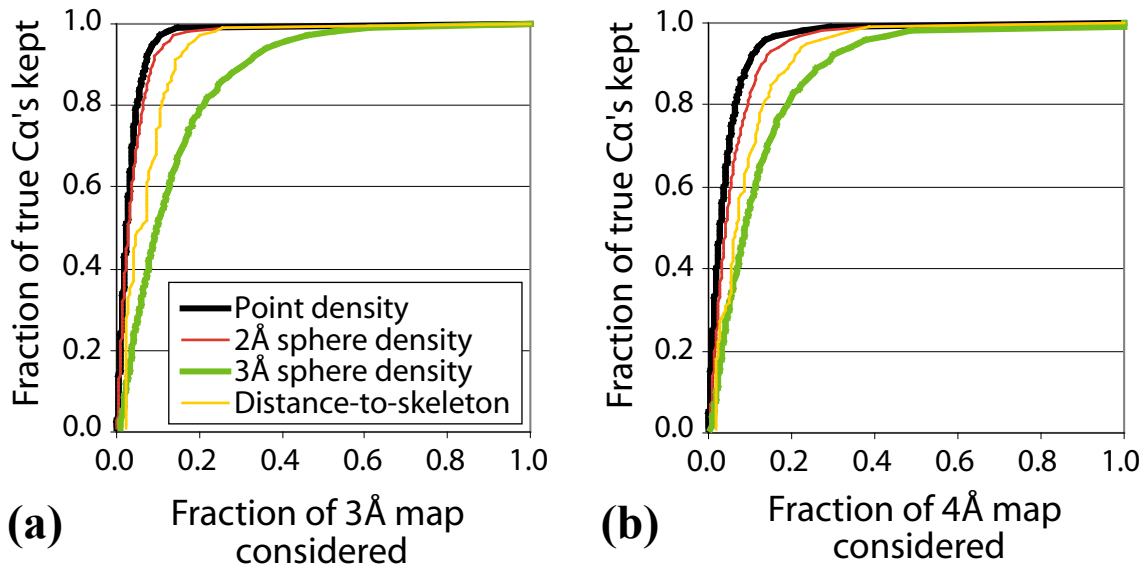


Figure 4.6 A comparison of four different filters for quickly eliminating some portion of the density map. Filter performance is compared on (a) 3Å and (b) 4Å resolution density maps.

4.4.2 First-pass filtering

A significant advantage of ACMI-SH over ACMI-FF is that my new approach allows me to filter out regions of the map that are very unlikely to have a Cα (the center of each template corresponds to a Cα), without needing to perform a computationally expensive rotational alignment. This section compares four different first-pass filters, all which are quickly computed.

The first three filters are based upon the observation that in density maps, especially poor-resolution maps, Cα locations correspond to the highest-density points in the map [73]. The first of these filters considers filtering points based on the individual point's density, while the other two consider the density sum in a 2 or 3Å radius around each point.

The fourth and final filter I test is based upon the *skeletonization* of the density map [40]. Skeletonization, similar to the medial axis transformation in computer vision, gradually "erodes" the density map until it is a narrow ribbon approximately tracing the protein's backbone and sidechains. I consider filtering each point based upon its distance to the closest skeleton point. This is the first-pass filter used by CAPRA [54] to eliminate points from the density map.

Figure 4.6 compares the performance of these four simple filters at both 3Å and 4Å resolution. These plots show, on the x-axis, the portion of the entire map I consider (sorted by my filter criteria), while the y-axis shows the fraction of true Cα locations included. For example, a point at coordinates $(0.2, 0.9)$ means a filter for which – at *some* threshold value – I look at only 20% of the density map and still find 90% of the true Cα locations. Somewhat surprisingly, the simplest filter,

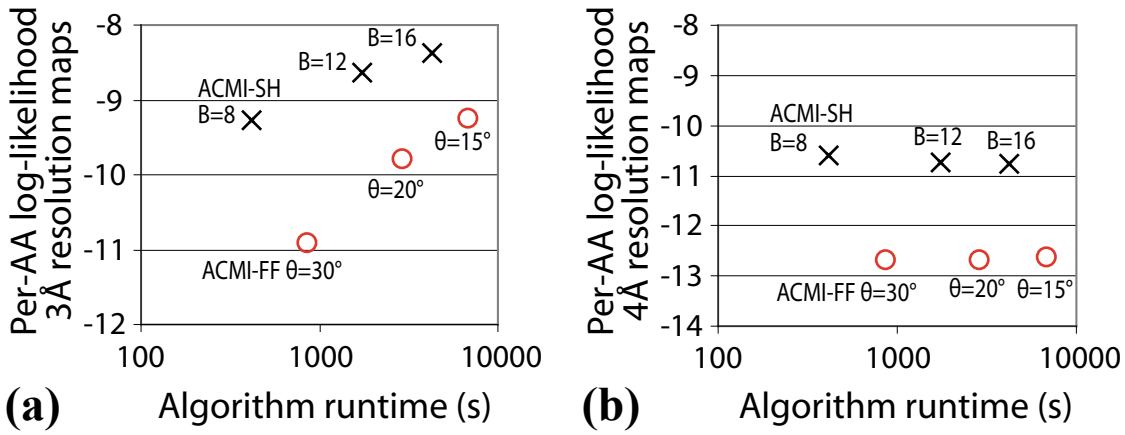


Figure 4.7 A comparison of ACMI-SH's and ACMI-FF's template matching on (a) 3Å and (b) 4Å resolution maps, in terms of average per-amino-acid log-likelihood of the true trace (higher values are better).

the point density, performs the best. The remainder of my experiments consider using the point-density as a first-pass filter, eliminating a conservative 80% of the density map from rotational search, and thus providing a 5-fold speedup in this (time-consuming) process.

4.4.3 Template matching

This section compares ACMI-FF and ACMI-SH's template-matching performance. I compare the performance of both algorithms as the angular sampling of each density template is varied. Given the sequences for each of the ten proteins in my testset, I consider searching for 10 randomly chosen amino-acids in each protein (100 amino acids total). For each amino acid, I find a minimum of 50 template pentapeptides with similar 5-mer sequences.

To test ACMI-FF, I cluster these pentapeptides based on the RMS deviation of their optimal alignment, and select a representative structure from each cluster. Further details are in the previous chapter. When testing my improved implementation (ACMI-SH), I perform no such clustering. Instead, I search the entire map for each of the 50+ templates. For ACMI-SH, I filter out all points below the 80th percentile density, assigning them some low probability.

Figure 4.7 compares the performance of ACMI-FF to ACMI-SH's search using spherical harmonics in both 3Å and 4Å density maps. In this plot, the x-axis measures the running time of the algorithm (in seconds), while the y-axis measures the per-amino-acid log-likelihood of the true solution. That is, given the deposited backbone model $\mathbf{B}^* = \{b_1^*, \ldots, b_K^*\}$, the y-axis measures

$$\mathrm{llk}(\mathbf{B}^*) = \frac{1}{K} \sum_{k=1}^{K} \log(\psi_k(b_k^*)) \tag{4.10}$$

Higher likelihoods are better; the more likely the true model, the more likely its structure will be recovered by inference.

It is interesting to note here that ACMI-SH, even at its lowest bandwidth limit, offers equal or better accuracy then the previous approach, in significantly less running time.

4.4.4 Comparison of protein models produced

In the previous chapter, I compared the performance of ACMI-FF+BP (that is, the sequential application of ACMI-FF and ACMI-BP on these ten maps to two other automated techniques specialized to low-resolution maps: Ioerger's TEXTAL and Terwilliger's RESOLVE, both described in Chapter 2.

I test ACMI-SH+BP on this same set of maps. Figure 4.8 compares the accuracy of the Cα model predicted by ACMI-SH+BP with that of ACMI-FF+BP, RESOLVE, and TEXTAL. Figures 4.8a and 4.8b show the average Cα RMS error and percentage of amino acids located over the ten structures. Figures 4.8c and 4.8d show scatterplots in which each individually solved electron-density map is a point. The x-axis indicates ACMI-FF+BP's error (or percent amino-acids identified); the y-axis shows the same metric for ACMI-SH+BP. Points in the shaded regions *below* the diagonal in Figure 4.8c and *above* the diagonal in Figure 4.8d correspond to maps in which ACMI-SH+BP produced the "better" interpretation (under the corresponding scoring criterion).

On these maps, ACMI-FF uses $\theta = 20^\circ$ angular discretization, while ACMI-SH is run with a bandwidth $B = 12$, and a filter that eliminated 80% of points based on the density of each point.

Here ACMI-SH+BP shows a clear improvement over all other approaches. Both Figures 4.7 and 4.8 show the greatest improvement in 4Å-resolution maps. Even with this improved accuracy, the running time of ACMI-SH is about 60% of that of ACMI-FF (see the middle dots in Figure 4.7).

The accuracy increase in using spherical harmonics likely comes from several different places. The increased efficiency allows a finer angular sampling: the bandwidth limit $B = 12$ is analogous to a 15° angular spacing. This increased efficiency also lets us search for each individual template – without clustering – which may help accuracy somewhat. Searching for a 5Å sphere, which captures the context of a particular amino acid (i.e., is an amino-acid typically on the surface or in the core of the protein?) may be improving the matching as well. Finally, band-limiting the signal, which throws out the highest-frequency components, may help eliminate noise from the density map.

4.5 Conclusions and future work

I described an improvement over ACMI-FF's template matching in three-dimensional template matching in electron-density maps. Chapter 3 describes a method using Fourier convolution to quickly search over all (x, y, z) coordinates for some rotation of a template. This chapter shows an alternative approach using the spherical-harmonic decomposition of a template to rapidly search all rotations of a template at a single (x, y, z) location. Unlike Fourier convolution, this method allows an initial filtering algorithm to reduce computational time by "masking out" locations in

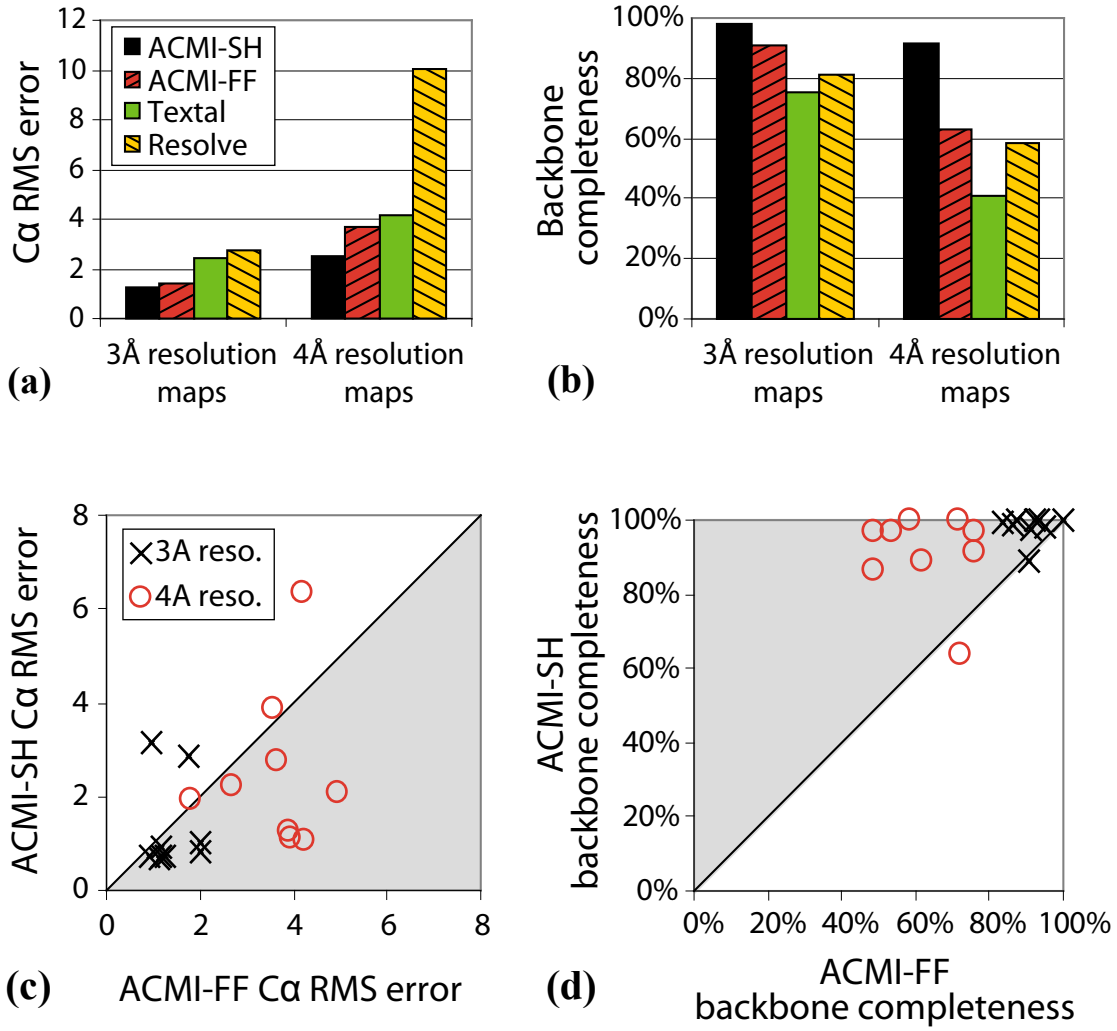


Figure 4.8 A comparison of ACMI-SH+BP's protein models with three other methods. (a) The average Cα RMS error and (b) percentage of amino acids located. Scatterplots compare ACMI-FF+BP's performance with ACMI-SH+BP's on (c) RMS error and (d) percentage of amino acids located. The shaded half of each scatterplot shows the region where ACMI-SH+BP is outperforming ACMI-FF+BP

the density map unlikely to contain an instance of any template. My improved template matching offers both improved efficiency and accuracy, compared to previous work, finding substantially better models in about 60% of the running time.

One goal for the future is further improvements in the first-pass filter. Even with the efficient fast rotation search presented here, rotationally aligning two regions of density is time-consuming.

Improved filtering would reduce the number of rotational alignments needed, significantly decreasing running time. One idea is to use a classifier based on the rotation-invariant representation suggested by Kondor [67].

Another future direction made possible by this work involves integrating this template searching and probabilistic inference (in ACMI-BP's Markov-field model). ACMI-SH makes it possible to efficiently search for a template at a single location. This suggests an approach where, initially, I search a small set of locations in some map. As inference in my model proceeds, locations that appear to be good candidates for Cα's may emerge. I could then search at these locations, in essence using the first few iterations of inference as a first-pass filter.

Chapter 5

Creating All-Atom Protein Models using Particle Filtering

The previous two chapters described a two-step method for constructing a coarse backbone model of a protein from an electron density map. ACMI-BP uses probabilistic inference to compute the probability distribution of the coordinates of each amino acid, given this density map. However, in constructing a protein-backbone model, ACMI-BP makes several simplifications, such as reducing each amino acid to a single atom (the Cα) and confining the locations to a coarse grid.

To address these limitations, this chapter introduces the use of a statistical sampling method called *particle filtering* (PF) [29] to construct all-atom protein models, by stepwise extension of a set of incomplete models drawn from a distribution computed by ACMI-BP. This results in a set of probability-weighted, all-atom protein models. The method interprets the density map by generating a number of distinct protein conformations consistent with the data.

I compare the single model that best matches the density map (without knowing the true solution) with the output of existing automated methods, on multiple sets of crystallographic data that required considerable human effort to solve. I also show that modeling the data with a *set* of structures, obtained from several particle-filtering runs, results in a better fit than using a single structure from a single particle-filtering run. Particle filtering enables the automated building of detailed atomic models for challenging protein crystal data, with a more realistic representation of conformational variation in the crystal. Some of the material in this chapter was previously published [23].

5.1 Shortcomings in ACMI-BP's model

The method introduced in Chapter 3, ACMI-BP, produces high-confidence backbone traces from a poor-quality density maps. Given a density map and the primary amino acid sequence of the protein contained in this map, ACMI-BP constructs a probabilistic model of the location of each Cα. Statistical inference on this model gives the most probable backbone trace *of the given sequence* in the electron-density map.

Formally, ACMI-BP's probabilistic inference returns, for each amino-acid i, the marginal probability distribution $\hat{p}_i(b_i)$ of that amino acid's Cα position. Previously, I computed the backbone trace $\mathbf{B}^* = \{b_1^*, \ldots, b_N^*\}$ (where b_i describes the position and rotation of amino-acid i) as the

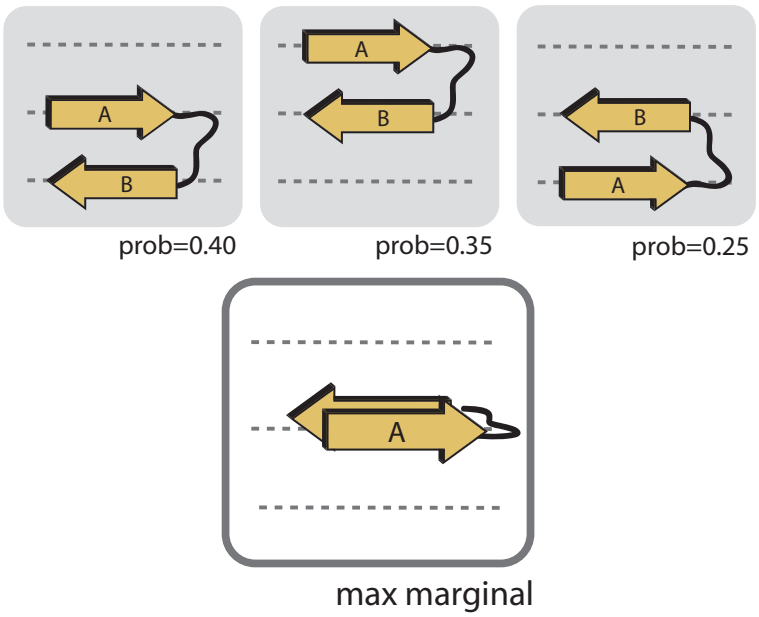


Figure 5.1 One case where a maximum-marginal trace may be undesirable. Suppose some protein takes any of the three conformations on the top row with the indicated probability. The maximum–marginal locations given this distribution places both sheets on top of each other, chain A with probability 0.4, and chain B with probability 0.6. This is clearly physically unrealistic.

position of each Cα that maximized ACMI-BP's belief,

$$b_i^* = \arg\max_{b_i} \hat{p}_i(b_i) \tag{5.1}$$

This chapter will use the variable $\mathbf{B}$ in place of the previous chapters' $\mathbf{U}$ to refer to a putative backbone model, and b_i instead of u_i to refer to the location of a single amino-acid i's Cα.

One obvious shortcoming in my previous approach is that biologists are interested in not just the position (b_i^*) of each Cα, but in the location of every atom in the protein. Naïvely, I could take ACMI-BP's most-probable backbone model, and simply attach the best-matching sidechain from a library of conformations to each of the model's Cα positions. In Section 5.4.2 I show that such a method works reasonably well.

However, this naïve approach does not address several other problems with ACMI-BP's backbone trace. ACMI-BP's marginal distributions are computed on a grid, which may lead to non-physical distances between residues when Cα's are placed on the nearest grid points. Additionally, ACMI-BP's inference is approximate, and errors due to these approximations may produce an incorrect backbone trace, with two adjacent residues located on opposite sides of the map. Figure 5.1 illustrates a third problem, dealing with independently selecting the position of each residue to maximize the marginal. Here, a protein that takes three possible backbone conformations with some non-negligible probability, leads to a physically unrealistic maximum-marginal trace.

Representing each amino acid's position as a distribution over the map is very expressive, and is easily able to model multiple backbone conformations. Simply returning the Cα position that maximizes the marginal ignores a lot of information. This chapter describes the application of particle filtering to "explain" the density map using multiple, physically feasible traces.

5.2 Particle-filtering overview

I will use a particle-filtering method called statistical importance resampling (SIR) [2, 29], which approximates the posterior probability distribution of a state sequence $x_{1:K} = \{x_1, \ldots, x_K\}$ given observations $y_{1:K}$ as the weighted sum of a finite number of point estimates $x^{(i)}_{1:K}$,

$$p(x_{1:K}|y_{1:K}) \approx \sum_{i=1}^{N} w_K^{(i)} \delta(x_{1:K} - x^{(i)}_{1:K}) \tag{5.2}$$

Here, i is the particle index, $w^{(i)}$ is particle i's weight, K is the number of states (in my application the number of amino acids), and δ is the Dirac delta[1] function. In my application, each x_k describes the position of every non-hydrogen atom in a single amino acid, and each y_k is some 3D region of density in the map.

In my work, the technical term "particle" refers to one specific 3D layout of all the non-hydrogen atoms in a contiguous subsequence of the protein (e.g., from amino acid 21 to 25). PF represents the distribution of some subsequence's layout using a set of distinct layouts for that subsequence, as illustrated in Figure 5.2 (in practice, several dozen to several hundred particles are used to represent a set *extensionally*).

At each iteration of particle filtering, I advance the extent of each particle by one amino acid. For example, given $x_{21:25} = \{x_{21}, \ldots .x_{25}\}$ the position of all atoms in amino acids 21 through 25 (I will use this shorthand notation for a particle throughout the chapter), PF samples the position of the next amino acid, in this case x_{26}. Ideally, particle filtering would sample these positions from the posterior distribution: the probability of x_{26}'s layout given the current particle and the map. SIR is based on the assumption that this posterior is difficult to sample directly, but easy to evaluate (up to proportionality). Given some other function (called the *importance function*) that approximates the posterior, particle filtering samples from this function instead, then uses the ratio of the posterior to the importance function to reweigh the particles.

To give an example of an importance function, particle-filtering applications often use the prior *conditional distribution* $p(x_k|x_{k-1})$ as the importance function. After sampling the data, y_k will be used to weight each particle. In my application, this is analogous to placing an amino acid's atoms using only the layout of the previous amino acid, then reweighing by how well it fits the map (however, this is *not* what my approach, described in Section 5.3, does).

I use a particle resampling step to address the problem of degeneracy in the particle ensemble [68]. As particles are extended, the variance of particle weights tends to increase, until there are

[1] This function is also known as the *impulse function*. It takes the value zero everywhere but the origin, where it is infinite; it is constrained such that $\int_{-\infty}^{+\infty} \delta(x)\, dx = 1$.

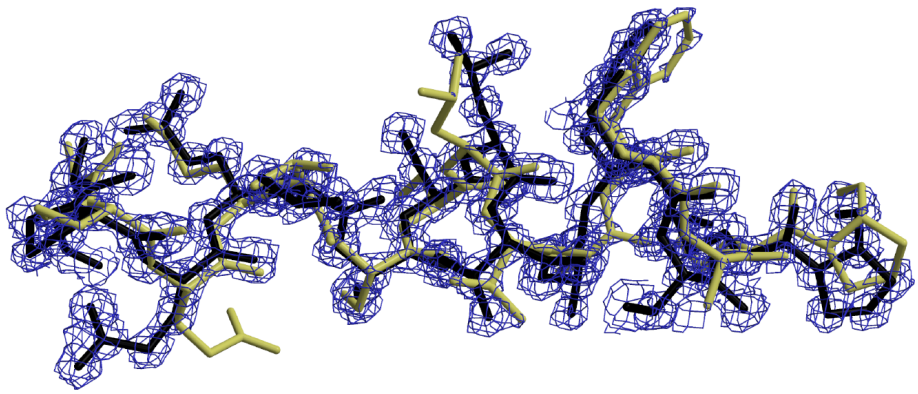


Figure 5.2 A pictorial look at particle filtering. I fit the density map shown using a set of particles (i.e. all-atom models). Here, only two particles are shown for clarity, one darker and one lighter.

few particles with non-negligible weights, and much effort is spent updating particles with little or no weight. To ameliorate this problem, an optional resampling step samples (with replacement) a new set of N particles at each iteration, with the probability of selecting a particle proportional to its weight. This ensures most particles remain on high-likelihood trajectories in state space.

One problem with resampling is that if particles are frequently resampled, the population of particles may become overly homogenized [2]. This is known as *sample impoverishment*. Others have developed methods to deal with this problem, including the *resample-move* algorithm [37] and regularization (replacing the Dirac δ with some other probability density estimate, such as a mixture-of-Gaussians) [87].

What makes SIR (and particle filtering methods in general) different from Markov chain Monte Carlo (MCMC) [88] is that MCMC is concerned with the stationary distribution of the Markov chain. In particle filtering, one is not concerned with convergence of the point estimates, rather, the distribution is simply modeled by the ensemble of particles, whether or not they converge. Variants of particle filtering improve the importance function through the use of auxiliary variables [96].

5.3 ACMI-PF's protein-particle model

I have developed ACMI-PF, an approach using particle filtering to construct an all-atom protein model. ACMI-PF wants to find the complete (all-atom) protein model $x_{1:N}$ (recall the variable x_i describes the location of all amino-acid i's non-hydrogen atoms) that best explains the observed electron-density map $\mathbf{y}$. To simplify, I parameterize x_k as a Cα location b_k (the same as b_i in Equation 5.1), and a sidechain placement s_k. The sidechain-placement variable contains the *3D location of every non-hydrogen sidechain atom* in amino-acid k, as well as the position of backbone atoms C, N, and O; s_k may contain the 3D coordinates of as many as 14 atoms.

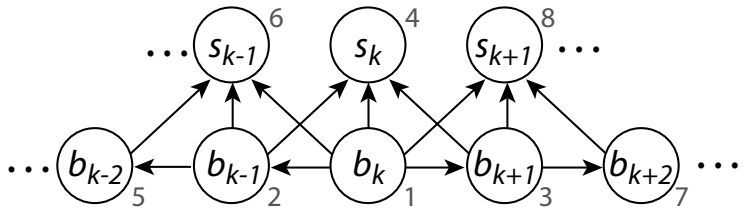


Figure 5.3 Conditional dependencies in sidechain (s_k) and Cα (b_k) layout. Numbers indicate the order in which labels are sampled, beginning with the middle Cα, and alternately moving forward and backward. Sidechain orientations are uniquely determined given $\langle b_{k-1}, b_k, b_{k+1} \rangle$.

Given this parameterization, the Markov process proceeds in two phases, alternating between placing: (a) Cα positions and (b) sidechain atoms (Figure 5.3). That is, an iteration of particle filtering first samples b_{k+1} (Cα of amino-acid $k+1$) given $b_k^{(i)}$ (alternately, growing a particle toward the N-terminus would sample b_{j-1} given $b_j^{(i)}$). This sampling makes use of ACMI-BP's inferred probability distribution of Cα $k+1$ locations (b_{k+1}) to place this amino acid in each particle. As the protein chain is grown, I decide whether to grow toward the N-terminus $(j-1)$ or C-terminus $(k+1)$ using ACMI-BP's marginal distributions, preferring to first sample amino acid's of which I am most sure ACMI-BP knows the location (details are shown in Section 5.3.3).

At this point, I have the locations of the alpha-carbon triplet $b_{k-1:k+1}^{(i)}$ for each particle. Given this, I next sample sidechain conformation s_k. Sidechain placement draws sidechain conformations (or *rotamers*) from a database of such conformations, and only considers these discrete conformations for s_k.

An overview of the complete algorithm appears in Algorithm 5.1. The following two subsections describe the backbone and sidechain steps, respectively, for a single particle.

5.3.1 Using ACMI-BP-computed marginals to place Cα's

In my algorithm's backbone step I want to sample the position of amino acid $k+1$'s Cα, b_{k+1} (or $(j-1)$'s Cα, b_{j-1}), given the incomplete Cα model from amino-acid j to amino-acid k, $b_{j:k}^{(i)}$, for each particle i. That is, I want to define a backbone sampling function $q(b_{k+1}|b_j^{(i)}, \ldots, b_k^{(i)}, y_k)$. The variable y_k here refers to a *region* of the electron density map in the neighborhood of amino-acid k.

Doucet *et al.* [29] defines the *optimal sampling function* as the conditional marginal distribution

$$q(b_{k+1}|b_j^{(i)}, \ldots, b_k^{(i)}, \mathbf{y}) = p(b_{k+1}|b_k^{(i)}, y_k) \tag{5.3}$$

While it is intractable to compute Equation 5.3 exactly, it is straightforward to estimate using ACMI-BP's Markov-field model

$$p(b_{k+1}|b_k^{(i)}, y_k) \propto \frac{p(b_k^{(i)}, b_{k+1}|y_k)}{p(b_k^{(i)}|y_k)} \tag{5.4}$$

Algorithm 5.1 ACMI-PF grows a protein model in two phases.

input: density map $\mathbf{y}$, amino-acid marginals $\hat{p}_k(b_k)$
output: set of protein models $x^{(i)}_{1:K}$ and weights $w^{(i)}_K$

`// start at some AA with high certainty about its location`
choose k such that $\hat{p}_k(b^{(i)}_k)$ has minimum entropy
foreach *particle* $i = 1 \dots N$ **do**
  choose $b^{(i)}_k$ at random from $\hat{p}_k(b^{(i)}_k)$
  $w^{(i)}_k \leftarrow 1/N$
end
foreach *residue* k **do**
  foreach *particle* $i = 1 \dots N$ **do**
    `// choose` b_{k+1} `(or` b_{k-1}`) given` $b^{(i)}_k$ `by subsampling` M `locations`
    $\{b^{*m}_{k+1}\} \leftarrow$ choose M samples from $\phi_{adj}(b^{(i)}_k, b_{k+1})$

    `// subsample weights proportional to Acmi-BP's belief`
    $w^{*m} \leftarrow$ belief $\hat{p}_i(b^{*m}_{k+1})$

    `// choose a subsample with probability proportional to belief`
    $b^{(i)}_{k+1} \leftarrow$ choose b^{*m}_{k+1} with probability $\propto w^{*m}$

    `// update particle weight as sum of subsample weights`
    $w^{(i)}_{k+1} \leftarrow w^{(i)}_k \cdot \sum_{m=1}^{M} w^{*m}$

    `// choose` s_k `given` $b^{(i)}_{k-1:k+1}$ `by subsampling` L `potential locations`
    $\{s^{*l}_k\} \leftarrow$ all sidechain conformations for amino-acid k

    `// subsample weights are probability of map given subsample`
    `//` $\mathrm{EDM}[b_k]$ `denotes region of density in the neighborhood of` b_k
    $p^{*l}_{null} \leftarrow$ prob $cc(s^{*l}_k, \mathrm{EDM}[b_k])$ occurred by chance

    `// choose a subsamples with probability proportional to weight`
    $s_k \leftarrow$ choose s^{*l}_k with probability $\propto 1/p^{*l}_{null} - 1$

    `// update particle weight as sum of subsample weights`
    $w^{(i)}_{k+1} \leftarrow w^{(i)}_k \cdot \sum_{l=1}^{L} 1/p^{*l}_{null} - 1$
  end
end

This numerator – the joint marginal of b_k and b_{k+1} – is easily computed in a Markov-field model. I compute this as the product of k and $(k+1)$'s belief, multiplied by the potential function between them, $\psi_{adj}(b_k, b_{k+1})$. Expanding this:

$$\begin{aligned} p(b_{k+1}|b_k^{(i)}, y_k) &\propto \frac{p(b_k^{(i)}, b_{k+1}|y_k)}{p(b_k^{(i)}|y_k)} \\ &\propto \frac{\hat{p}_k(b_k^{(i)}) \cdot \psi_{adj}(b_k^{(i)}, b_{k+1}) \cdot \hat{p}_{k+1}(b_{k+1})}{\hat{p}_k(b_k^{(i)})} \\ &\propto \hat{p}_{k+1}(b_{k+1}) \cdot \psi_{adj}(b_k^{(i)}, b_{k+1}) \end{aligned} \tag{5.5}$$

Here, $\hat{p}_{k+1}(b_{k+1})$ is the ACMI-BP-computed marginal distribution for amino-acid $k+1$ (the dependence of $\hat{p}_k$ and $\hat{p}_{k+1}$ on y_k has been dropped for readability).

That is, what I sample – my importance function – is the location of Cα $k+1$ from the product of (a) $k+1$'s marginal distribution and (b) the adjacency potential between Cα k and Cα $k+1$. A (mostly) accurate way of thinking about this is – given that I have the location of the ith Cα (at b_i) – I want to determine likely places for the $(i+1)$th Cα. I do this by looking 3.8Å away (I know Cα's are separated by this distance), and use ACMI-BP's distribution of Cα $(i+1)$'s location to help me choose a direction to take.

Doucet's optimal sampling function has a corresponding weight update

$$w_{k+1}^i \propto w_k^i \times \int p(y_{k+1}|b_{k+1}, b_k^{(i)})\, db_k \tag{5.6}$$

This integral, too, is intractable to compute exactly, but can also be approximated using ACMI-BP's marginals,

$$w_{k+1}^i \propto w_k^i \times \int \hat{p}_{k+1}(b_{k+1}) \cdot \psi_{adj}(b_k^{(i)}, b_{k+1})\, db_k \tag{5.7}$$

Notice that the function over which I integrate (Equation 5.7) is exactly the function from which I want to sample (Equation 5.5). This suggests a sampling approach to the problem of choosing location of Cα $k+1$ and reweighing each particle. This sampling approach is illustrated pictorially in Figure 5.4.

I sample M *potential* Cα locations using only the adjacency potential between k and $k+1$, which models the allowable conformations between two adjacent Cα's. I will refer to each of these M "subsamples" (each particle is a sample, and I take M samples per particle) using the notation b_{k+1}^{*m}, $m = 1, \ldots, M$. That is, I sample:

$$b_{k+1}^{*m} \sim \psi_{adj}(b_k^{(i)}, b_{k+1}) \tag{5.8}$$

Keep in mind that $b_k^{(i)}$ is "bound": I sampled its location (for this specific particle) previously.

I assign each sample a *weight*, the (approximate) marginal probability $\hat{p}_{k+1}(b_{k+1}^{*m})$ at each of these sampled locations. Since each subsample b_{k+1}^{*m} is continuously valued, and ACMI-BP's beliefs $\hat{p}_{k+1}$ are defined on a grid covering the unit cell, I use nearest-neighbor interpolation. That is,

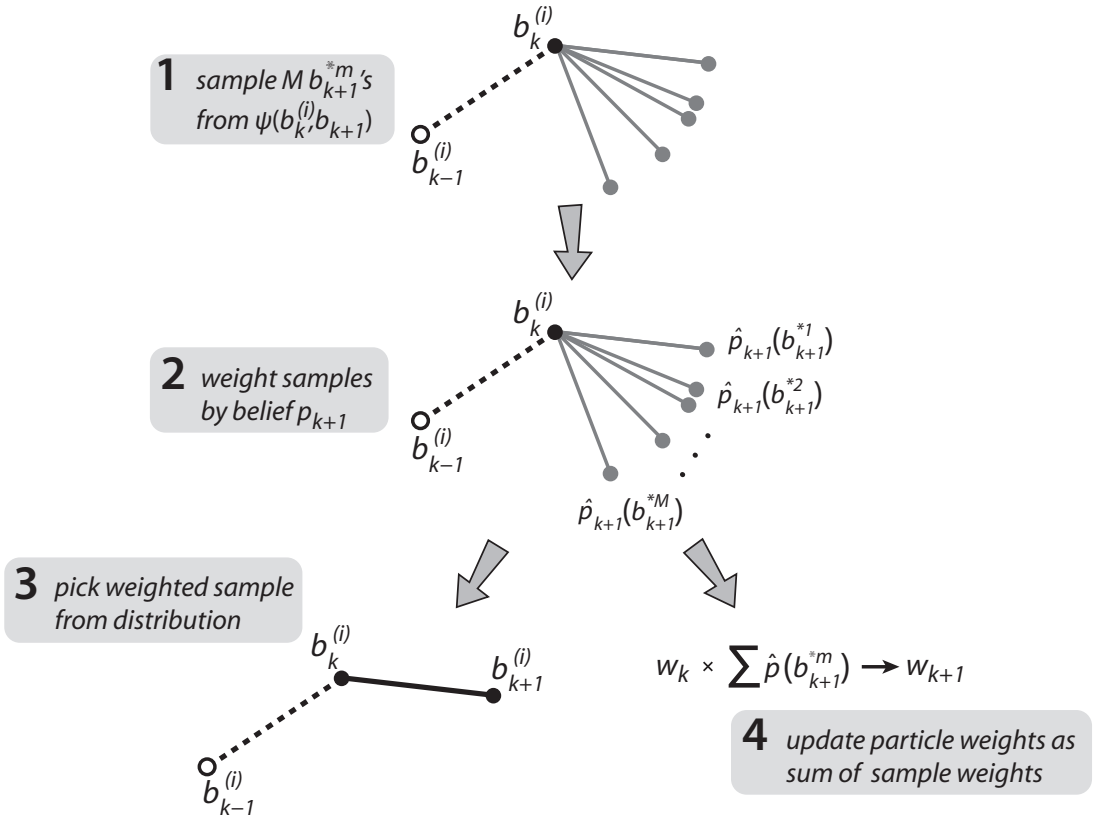


Figure 5.4 An overview of the backbone forward-sampling step. Given positions b_{k-1} and b_k, I sample M positions for b_{k+1} using the empirically-derived distribution of Cα–Cα–Cα pseudoangles. Each potential b_{k+1} is weighted by the belief $\hat{p}(b_{k+1}^{*m}|\mathbf{y})$. I choose a single location from this distribution; the particle weight is multiplied by the *sum* of these weights.

I take the value of ACMI-BP's belief at each subsample to be the belief at the closest grid point over which beliefs are defined.

I then choose a sample from this weighted distribution. This provides a good approximation to sampling from Equation 5.5; the approximation error approaches 0 as $M \rightarrow \infty$ (my experiments typically use $M = 10^5$). Finally, I reweight the particle as the average of weights of *all* M samples considered:

$$w_{k+1}^i \propto w_k^i \times \frac{1}{M} \times \sum_{m=1}^{M} \hat{p}_{k+1}(b_{k+1}^{*m}) \cdot \psi_{adj}(b_k^{(i)}, b_{k+1}^{*m}) \tag{5.9}$$

This sum approximates the integral in Equation 5.7. Since we only care about each particle's weight up to proportionality (weights are normalized to sum to unity over all particles), typically the $1/M$ term is dropped, and particles are reweighted using the *sum* of sample weights.

This process, in which I consider M potential Cα locations, is repeated for every particle in the particle filter for each Cα in the protein.

5.3.2 Using sidechain templates to sample sidechains

Once my particle filter has placed Cα's $(k-1)$, k, and $(k+1)$ at 3D locations $b^{(i)}_{k-1:k+1}$, it is ready to place all the sidechain atoms in amino-acid k. I denote the position of these sidechain atoms s_k. Given the primary amino acid sequence near k (specifically, the 5-mer centered at k), I draw a set of previously observed sidechain conformations from a nonredundant subset of the Protein Data Bank (PDB)[2] [115]. To reduce computational complexity, I only let s_k take one of these previously observed sidechain conformations (a flexible sidechain model, like the one described in Chapter 6, would require excessive computational time, as each sidechain must be placed for each of the N particles). Therefore, s_k consists of (a) an index into a database of known sidechain 3D structures and (b) a rotation.

To further simplify, I construct all of my sidechain templates to model the position of every atom from the *previous* $(k-1)$ amino acid's Cα to the *next* $(k+1)$ amino acid's Cα. That is, the sidechain template contains three consecutive alpha carbons. Then, given three consecutive Cα's $b^{(i)}_{k-1:k+1}$ already placed in some particle, the orientation of sidechain s_k is uniquely determined by aligning the three Cα's in the sidechain template to the particle's backbone positions $b^{(i)}_{k-1:k+1}$ (for a cartoon representation of this alignment, see step 2 in Figure 5.5).

Figure 5.5 shows the process of choosing a sidechain conformation for a single particle i. Sidechain sampling and particle reweighing is quite similar to the Cα placement in the previous section. A key difference is that sidechain placement cannot take advantage of ACMI-BP's marginal distribution, as ACMI-BP's probability distributions have marginalized away sidechain conformations. Instead, the probability of a sidechain is calculated on-the-fly using the cross-correlation between a sidechain template's density and the electron density in a region around b_k in the electron-density map. That is, for each sidechain subsample (i.e., conformation) s^{*l}_k, $l \in \{1, \ldots, L\}$, I compute the correlation coefficient between the conformation and the map

$$CC^l = \text{cross-correlation}(s^{*l}_k, \text{EDM}[b^{(i)}_k])$$

$\text{EDM}[b_k]$ denotes a region of density in the neighborhood of b_k. The "neighborhood" taken is a region of density corresponding to each sidechain template s^{*l}_k: I construct a mask containing all grid points within 2.5Å of any atom in s^{*l}_k, if it were centered at b_k.

To assign a probability $p(\text{EDM}[b^{(i)}_k]|s_k)$ to each sidechain conformation, I compute the probability that a cross-correlation value was not generated by chance. That is, I assume that the distribution of the cross correlation of two random functions is normally distributed with mean μ and variance σ^2. I learn these parameters by computing correlation coefficients between randomly sampled locations in the given map. Given some cross correlation x_c, I compute the expected probability that I would see score x_c or higher by random chance,

$$p_{null}(x_c) = P(X \geq x_c; \mu, \sigma^2) = 1 - \Phi(x_c - \mu/\sigma) \tag{5.10}$$

[2] As in previous chapters, proteins from my testset are removed from this database prior to testing.

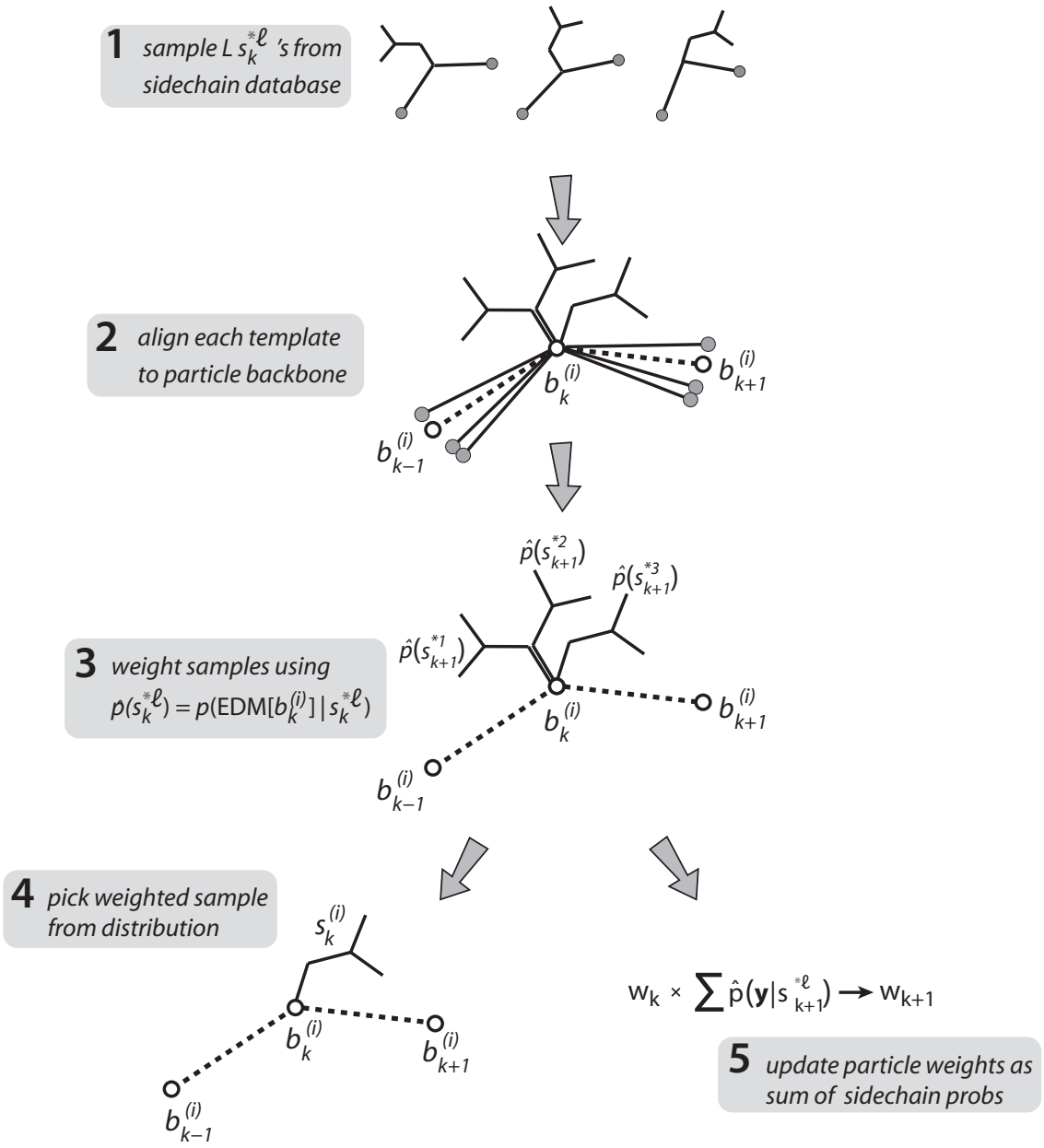


Figure 5.5 An overview of the sidechain sampling step. Given positions $b_{k-1:k+1}$, I consider L sidechain conformations s_k^{*l}. Each potential conformation is weighted by the probability of the map given the sidechain conformation, as given in Equation 5.11. I choose a sidechain from this distribution; the particle weight is multiplied by the *sum* of these weights.

Here, $\Phi(x)$ is the normal cumulative distribution function. The probability of a particular sidechain conformation is then

$$p(\mathrm{EDM}[b_k^{(i)}]|s_k^{*l}) \propto (1/p_{null}) - 1 \tag{5.11}$$

Since I draw sidechain conformations from the distribution of all solved structures, I assume a uniform prior distribution on sidechain conformations, so $p(s_k^{*l}|\,\mathrm{EDM}[b_k^{(i)}]) \propto p(\mathrm{EDM}[b_k^{(i)}]|s_k^{*l})$.

As illustrated in Figure 5.5, sidechain sampling uses a method similar to the backbone sampling of the previous section. I consider extending a particle by each of these L sidechain subsamples $\{s_k^{*1}, \ldots, s_k^{*L}\}$ from my sidechain database. After computing the cross correlations between each sidechain and the density map around b_k, each sidechain conformation is weighted by the probability in Equation 5.11. I choose a single conformation at random from this weighted distribution, and update a particle's weight by the *sum* of weights of all the sidechain conformations I considered.

My model takes into account the complete previous trajectory $x_{j:k-1}$ when placing sidechain s_k. If any atom in sidechain s_k overlaps a previously placed atom, the particle weight is set to zero. This prevents two atoms from occupying the same 3D space; it also prevents two symmetric copies of a protein from overlapping.

Using particle filtering, one amino acid k is placed for particle i, the position of k is never updated. Early "bad decisions," which are not retracted, are overcome by having an ensemble. If a poor decision is made, when it later becomes evident, the corresponding particle is assigned low weight, and is eventually eliminated through resampling. Note that this can still lead to trouble if the correct early choice is very unlikely (that is, involves a very low-probability step).

5.3.3 Sampling order

Every particle in the population samples Cα's in the same order (this is necessary for resampling to work properly). The order in which Cα's are assigned also makes use of ACMI-BP's distributions. For every particle, I begin sampling in the amino-acid k whose marginal distribution has the lowest entropy H_k:

$$H_k = -\sum_{x_k} \hat{p}_k(x_k) \log \hat{p}_k(x_k) \tag{5.12}$$

The amino acid with lowest entropy corresponds to the amino acid which ACMI-BP is most sure it knows the location. I use a "soft" minimum; that is, I choose an amino-acid k with probability proportional to $\exp(-H_k)$. This ensures the order in which amino acids are placed is different in multiple executions of the algorithm. The direction I sample at each iteration (i.e. toward the N- or C-terminus) is also decided by the entropy of the marginal distributions (again, using a soft minimum).

Although a single iteration of particle filtering generates many models, these particles typically express little diversity (see the discussion of *sample impoverishment* in Section 5.2), especially in the amino acids placed during the first half of the algorithm. This is due to the significant number of resampling steps required during particle filtering: A protein 1000 amino acids in length takes 2000 sampling steps and resampling is typically necessary every 2-4 steps. Furthermore, strict constraints on interatomic distances and angles make resampling from a kernelized density

Algorithm 5.2 ACMI-PF generates multiple models through multiple independent runs.

input: density map **y**, amino-acid marginals $\hat{p}_k(b_k)$, number of models to generate R
output: set of *independently sampled protein models* $x_{1:R}$

foreach *particle-filtering run* $r = 1, \ldots, R$ **do**
    `// An independent particle-filtering run generates`
    `//     particles P with weights W`
    $(\mathbf{P}, \mathbf{W}) \leftarrow \text{PF}(\mathbf{y}, \hat{p}_k(b_k))$
    `// Choose the particle with greatest weight`
    $i_{max} \leftarrow \arg\max_i (w_i)$
    $x_r \leftarrow p_{i_{max}}$
end

estimate (a common method [87] for combating particle homogeneity) non-trivial. That is, if I consider resampling each atom's location from a Gaussian centered on the original particle, I will likely get a model in which these distance and angular constraints have been violated.

Therefore, to generate multiple protein models, I consider independent executions of particle filtering, taking the single highest-weight particle from each execution [81]. A high-level overview is shown in Algorithm 5.2. Using independent runs also mean that each execution samples amino acids in a different order, which may offer some advantage, introducing additional diversity in the models. Finally, a postprocessing step refines each of the structures for 10 iterations using REFMAC5 [86].

5.4 Experiments

I use ten experimentally phased electron-density maps to test ACMI-PF (The results section will refer to the sequential application of ACMI-SH, ACMI-BP, and ACMI-PF as simply "ACMI-PF"). The data is provided by the Center for Eukaryotic Structural Genomics (CESG) at UW–Madison, as MTZ files with experimental intensities and with the initial phasing (typically obtained using SAD or MAD) available to the crystallographer at the start of model-building.

Details of the dataset are shown in Section A.2, in Appendix A. I remove maps in my testset from this non-redundant PDB subset before testing. These structures have been previously solved and deposited to the PDB, enabling a direct comparison with the final refined model. All ten required a great deal (several days to several months, depending on the map) of human effort to build and refine the final atomic model.

5.4.1 Methodology

Models in ACMI-PF are built in three phases: first, ACMI-SH computes prior distributions (Chapter 4), then ACMI-BP (Chapter 3) computes posterior distributions for each Cα, and finally ACMI-PF constructs all-atom models using particle filtering. Where available, ACMI-SH

used information about location of selenium atom peaks as a soft constraint on the positions of corresponding methionine residues in the computation of priors (see Section 3.2.3 for a complete description of this process). I run particle filtering ten times; in each run, I return the single highest-weight model, producing a total of ten ACMI-PF protein models. I refine each predicted model for 10 iterations using REFMAC5 [86], with no modification or added solvent. The first phase is the most computationally expensive, but is efficiently divided across multiple processors. I used a cluster of Linux machines for most calculations. Computation time varied depending on the size of the protein and of the asymmetric unit; the entire process took several days to a week of CPU time on ten processors for large proteins.

I compare ACMI-PF to four different approaches on the same ten density maps. To test the utility of the particle-filtering method for building all-atom models, I use the structure that results from independently placing the best matching sidechain on each Cα predicted by ACMI-BP, an algorithm which I term ACMI-NAÏVE. The other three approaches are the commonly used density-map interpretation algorithms ARP/wARP [84], TEXTAL [55], and RESOLVE [110]. Refinement for all algorithms uses the same protocol as ACMI-PF, refining the predicted models for 10 iterations in REFMAC5 (ARP/wARP, which integrates refinement and model-building, was not further refined).

To assess the prediction quality of each algorithm, I consider three different performance metrics: (a) backbone completeness, (b) sidechain identification, and (c) R factor. The first metric compares the predicted model to the final crystallographer-determined model, and counts – over each predicted chain – the fraction of Cα's placed within 2Å of *some* Cα chain in the PDB-deposited model. The distance of 2Å is chosen as a cutoff distance that is approximately half of the distance between adjacent alpha carbons. The second measure counts the fraction of Cα's both correctly placed within 2Å *and* whose sidechain type matches the PDB-deposited structure (as described in Chapter 2, all three alternate approaches – which separate placing the backbone and aligning it to the input sequence – often have difficulty identifying sidechains in poor-quality maps, returning long stretches of unidentified residues).

My final measure, the R factor, is a statistical residual measure of the deviation between the reflection intensities predicted by the model and those experimentally measured. A lower R factor indicates a better model. The R factor is computed using only peptide atoms, not water molecules or ARP/wARP's pseudoatoms. The comparison uses the so-called *free R factor* (R_{free}) [10], which is based on reflections that were not used in refinement protocol to avoid overfitting.

5.4.2 ACMI-NAÏVE versus ACMI-PF

In this section, I compare protein models produced by ACMI-PF to those produced by ACMI-NAÏVE. The key advantage of particle filtering is the ability to produce multiple protein structures using ensembles of particles. Since the density map is an average over many molecules of the protein in the crystal, it is natural to use multiple conformations to model this data. There is evidence that a single conformation is insufficient to model the electron density of proteins [11, 22, 35, 74]. As a comparison, I take the ACMI-NAÏVE approach that uses the maximum-marginal trace to produce a single model.

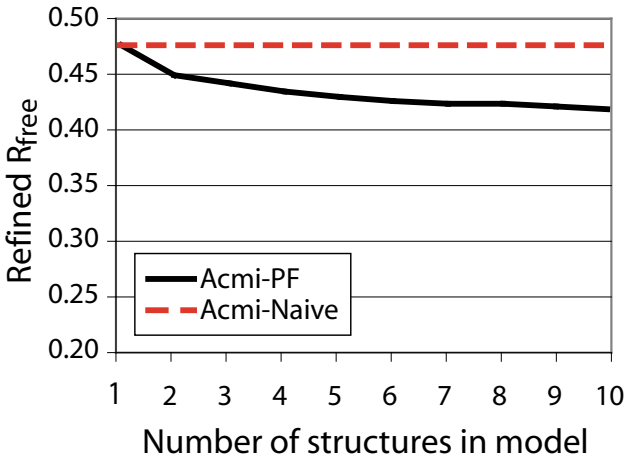


Figure 5.6 A comparison of the R_{free} of ACMI-NAÏVE and ACMI-PF, as the number of protein models produced varies. Multiple models are produced by independent ACMI-PF runs (ACMI-NAÏVE only produces a single model). Since R_{free} in deposited structures is typically 0.20-0.25, we use 0.20 as the lowest value on the y-axis.

I used ACMI-PF to generate multiple physically feasible models, by performing ten different ACMI-PF runs of 100 particles each. Each run sampled amino acids in a different order; the amino acids whose belief had lowest entropy (i.e., the position had greatest confidence) were stochastically preferred. Figure 5.6 summarizes these results. In this plot, the y-axis shows the average (over the ten density maps) R_{free} of the final refined model, while the x-axis indicates the number of ACMI-PF runs. This plot shows that a single ACMI-PF model has an R_{free} approximately equal to the R_{free} of ACMI-NAÏVE. Model completeness is also very close between the two (data not shown). Moreover, as additional structures are added to the model from multiple ACMI-PF runs, the average R_{free} decreases. The plot shows ACMI-NAÏVE's model as a straight line, since there is no mechanism to generate multiple conformations.

Finally, individual models in ACMI-PF also offer additional advantages over ACMI-NAÏVE. Comparing the *single* ACMI-PF model (of ten generated) with lowest ${R_{work}}^3$ to ACMI-NAÏVE's model shows that particle filter produces fewer chains on average (28 versus 10) and lower all-atom RMS error (1.60Å versus 1.72Å). In all ten maps in the test set, this trend held: ACMI-PF's best model contained fewer predicted chains and lower all-atom RMS error than ACMI-NAÏVE. Additionally, the structures returned by particle filtering are *physically feasible*, with no overlapping sidechains, and no invalid bond lengths.

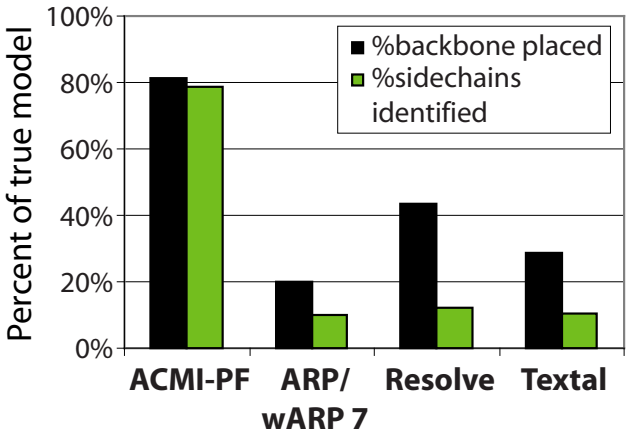


Figure 5.7 A comparison of ACMI-PF and ACMI-NAÏVE to three other automatic interpretation methods in terms of average backbone completeness and sidechain identification.

5.4.3 Comparison to other algorithms

I further compare the models produced by particle filtering on the ten maps to those produced by three other popular methods for automatic density-map interpretation. I use two well-established lower-resolution algorithms, TEXTAL and RESOLVE, and also include the atom-based ARP/wARP, even though most of the test maps are outside of its recommended resolution limit.

Figure 5.7 compares all four methods in terms of backbone completeness and sidechain identification, averaged over all ten structures. To provide a fair comparison, I compute completeness of a single ACMI-PF structure (of the ten produced). The ACMI-PF model chosen was that with the lowest refined R_{work}.

Under both of these metrics, ACMI-PF locates a much greater fraction of the protein than the other approaches. ACMI-PF performs particularly well at sidechain identification, correctly identifying close to 80% of sidechains over these ten poor-quality maps. The least accurate model that ACMI-PF generated (for protein 2AB1) had 62% backbone completeness and 55% sidechain identification. In contrast, the three comparison methods all had at least five structures with less than 40% backbone completeness and at least eight structures with less than 20% sidechain identification.

Additionally, the scatterplots in Figure 5.8 compare the R_{free} of ACMI-PF's *complete* (10-structure) model to each of the three alternative approaches, for each density map. Any point below the diagonal corresponds to a map for which ACMI-PF's solution has a lower (i.e., better) R_{free}. These plots show that for all but one map ACMI-PF's solution has the lowest R factor. The singular exception for which ARP/wARP has a lower R factor is protein 2NXF, a high (1.9Å) resolution but poorly phased density map in which ARP/wARP automatically traces 96%, while ACMI-PF's best model (that is, with lowest refined R_{work}) correctly predicts only 74%. The collection of

[3] R_{work} is the R factor on the "training set"; that is, the 90-95% of reflections used in refinement.

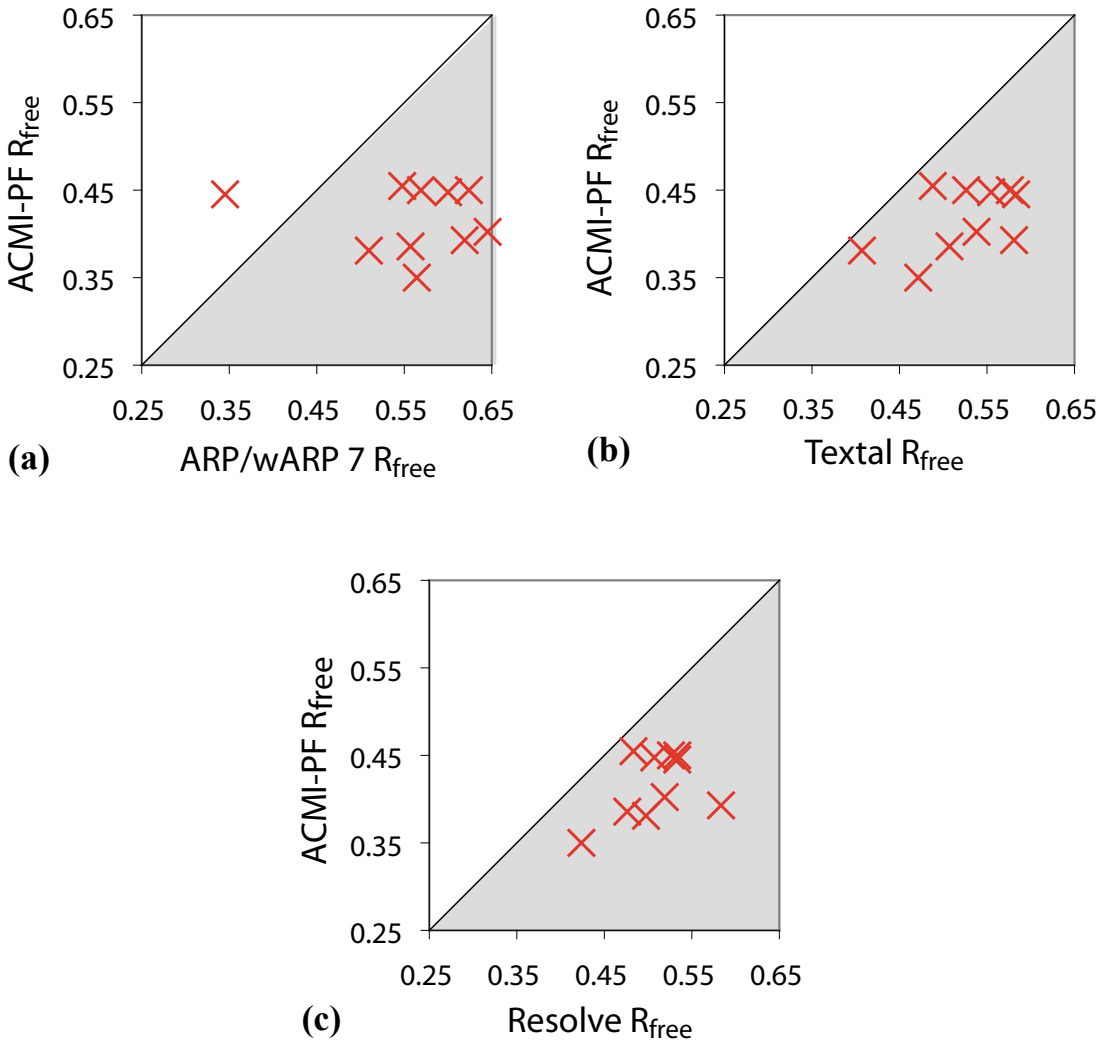


Figure 5.8 A comparison of the free R factor of ACMI-PF's interpretation for each of the ten maps versus (a) ARP/wARP, (b) TEXTAL, and (c) RESOLVE. The scatterplots show each interpreted map as a point, where the x-axis measures the R_{free} of ACMI-PF and the y-axis the alternative approach. The shaded half of each scatterplot (below the diagonal) shows the region where ACMI-PF produced the better model.

results illustrates both the limitations and the advantages of ACMI-PF: it is consistently superior at interpretation of poorly phased, lower resolution maps, while an iterative phase-improvement algorithm like ARP/wARP may be better suited for a poorly phased but higher-resolution data.

5.5 Iterative phase improvement with ACMI-PF

This section details some initial experiments describing the use of ACMI-PF in an iterative manner, where the quality of the maps (and hopefully the resulting models) are improved in each iteration. This iterative approach is similar to that of ARP/wARP, which iterates between improving map-quality and constructing a protein model to more accurately interpret poor-quality density maps.

5.5.1 The phase problem

The phase problem, briefly introduced in Chapter 2, comes about because electron-density maps are constructed as a function of two variables: (a) reflection intensities I, and (b) reflection phases Φ. Only the reflection intensities are measurable by X-ray crystallography; the phases are unmeasurable (or "lost") by the crystallographic process. A crystallographer must instead estimate these phases, using experimental or computational methods. Experimental methods for phase determination – for example, SAD [46], MAD [44], or MIR [8] – make use of scattering properties of heavy atoms to produce an initial estimate of phases, while computational methods use the phases from a protein with (presumed) similar 3D structure as a surrogate for the true phases.

The problem that arises is that these initial experimental phases are typically inaccurate, and produce a noisy density map. Figure 2.6 shows the effects of phase error on density-map quality. To deal with poorly phased maps, crystallographers typically use an iterative approach, like that shown in Figure 5.9. Beginning with measured intensities I_{obs} and experimentally determined phases Ψ_{exp}, a crystallographer computes the electron-density map (as the Fourier transform of these complex-valued reflections). Using ACMI-PF or some other method, the crystallographer builds a partial atomic model of the protein contained in the density map. From a partial model, one computes the *expected* intensities and phases, I_{calc} and Ψ_{calc}, given the atomic model.

Although the initial atomic model may be very incomplete, often these calculated phases are more accurate than the experimentally determined phases. This suggests an iterative approach in which the phases calculated from the atomic model are used to improve the density map. Using the improved density map a crystallographer (or ACMI-PF) may produce a more complete atomic model, and generate more accurate calculated phases. This iteration may continue until no additional improvement is possible. In practice, the calculated phases and experimental phases improve the model by either: (a) using the calculated phases only or (b) combining the two through a statistical weighing function, like that of SIGMA-A [45].

A more complete discussion of the phase problem can be found in any crystallography textbook (e.g., [97]).

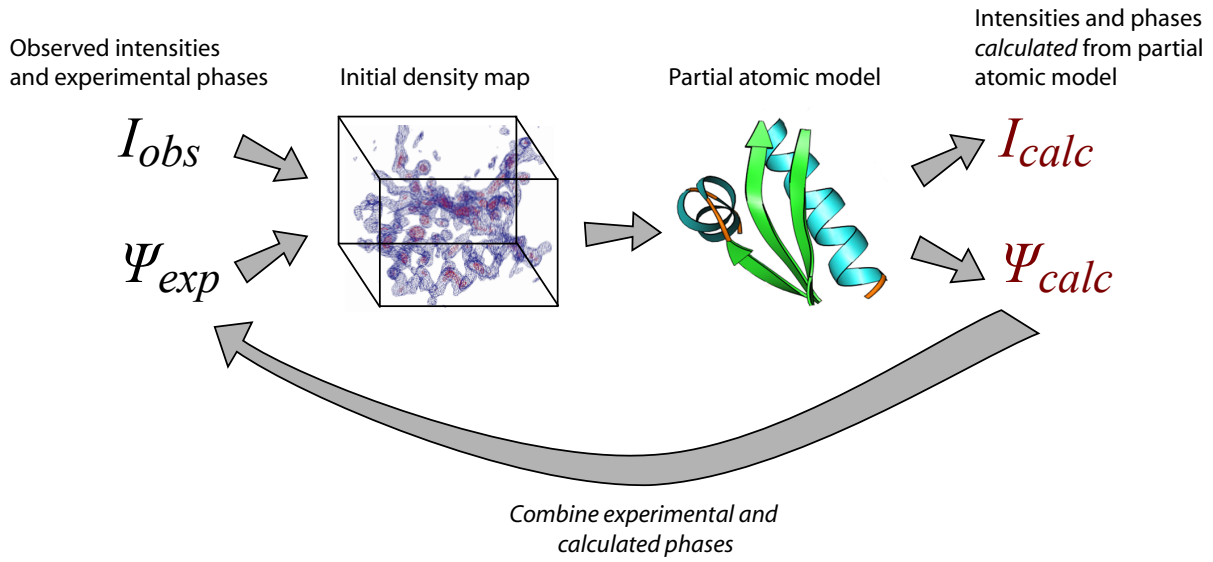


Figure 5.9 An overview of iterative phase improvement in crystallographic density maps. A crystallographer begins with measured intensities I_{obs} and experimentally determined phases Ψ_{exp}, from which a density map is calculated. A partial model is constructed from this map, and one computes the *expected* intensities and phases given the model (I_{calc} and Ψ_{calc}). The iterative component uses the idea that the calculated phases (or some combination of calculated and experimental phases) may produce a better estimate of the true phases, from which a better model may be constructed.

5.5.2 Using ACMI-PF for phase improvement

The first question I want to answer is: do the calculated phases from ACMI-PF lead to reduced phase errors? Unfortunately, I do not know the true phases of the density maps in our testset; however, I can use the calculated phases of the final deposited structure as a reasonable approximation. In this section I compare – for each of the ten testset maps – the *mean phase error* of the experimental phases to the calculated phases from ACMI-PF.

Figure 5.10 compares the experimental phase error to the phase error of ACMI-PF's calculated phases. The scatterplot shows each interpreted map as a point, with the x-axis the mean phase error of experimental phases and the y-axis the mean phase error of ACMI-PF's calculated phases. For all ten maps, the phase error is lower using ACMI-PF. This reduced phase error is especially evident in poorly phased maps: in one map, phase error is reduced from 66° to 42°; in another it is reduced from 66° to 56°. This seems to indicate that ACMI-PF is suitable for iterative phase improvement.

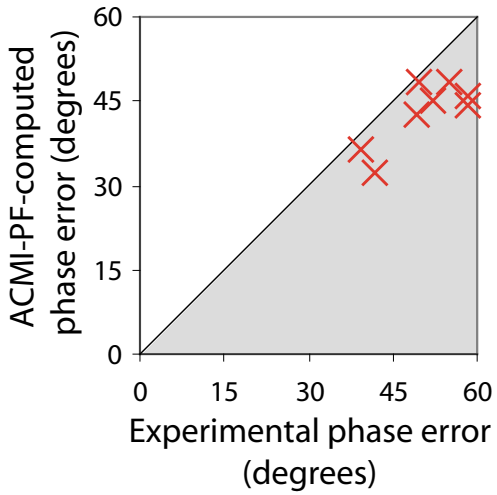


Figure 5.10 A scatterplot comparing the error of ACMI-PF's calculated phases with the error of the experimentally estimated phases. The x-axis indicates the mean phase error of experimental phases, while the y-axis indicates the mean phase error of ACMI-PF's calculated phases. The shaded region shows maps for which ACMI-PF's calculated phases have a reduced phase error.

Additionally, Table 5.1 shows that using the calculated phases from ACMI-PF's 10-structure model also yields reduced phase error compared to: (a) the calculated phases from the best ACMI-PF protein structure and (b) the SIGMA-A-weighted phases, which probabilistically combine calculated phases from the best ACMI-PF protein structure with experimental phases [45]. For all ten maps, the phase error is minimized using calculated phases from all ten ACMI-PF structures.

This indicates that the calculated phases from multiple PF-generated structures would be valuable for iterative phase improvement of the density map. I believe a key reason for this result is that particle filtering occasionally makes mistakes when tracing the main chain in poor-quality regions. However, it is unlikely for multiple particles to repeat the same mistake. Therefore, using multiple models, the mistakes average out in the ensemble, producing a lower R factor, and improved modeling of phases.

5.5.3 Multiple iterations of ACMI-PF

Finally, I compute improved density maps using ACMI-PF's calculated phases and run another iteration of the complete ACMI pipeline (ACMI-SH, ACMI-BP, and ACMI-PF). To reduce computation time, I only run a second ACMI iteration on the smallest five density-maps from our testset. I compare the initial ACMI-PF results (*iteration 1*) with the results on the improved density maps (*iteration 2*).

The results on these five density maps are summarized in Figure 5.11. I compare ACMI-PF's iteration-1 and iteration-2 models using free R factor (Figure 5.11a) and completeness of the

Table 5.1 The use of multiple protein structures reduces phase errors more than a single structure. For all ten testset maps, the phase error is lower using the calculated phases from ACMI-PF's 10-structure model (fifth column) than the: experimental phases (second column), calculated phases from the single best ACMI-PF structure (third column), or the combined calculated and experimental phases (fourth column).

PDB id	Experimental phase error	Single-structure phase error	Single-structure SIGMA-A-combined phase error	10-structure phase error
2NXF	58°	58°	52°	**46°**
2Q7A	49°	51°	46°	**43°**
XXXX[a]	54°	56°	50°	**45°**
1XRI	39°	46°	37°	**36°**
1ZTP	42°	39°	34°	**32°**
1Y0Z	58°	52°	48°	**44°**
2A3Q	66°	49°	47°	**42°**
2IFU	50°	54°	53°	**48°**
2BDU	55°	59°	58°	**48°**
2AB1	66°	64°	59°	**56°**
average	*54°*	*53°*	*49°*	*44°*

[a] PDB file not yet released.

single best PF structure (Figure 5.11b). The scatterplots show each interpreted map as a point, with the shaded regions indicating maps for which the second iteration produced a better model. Unfortunately, the results in this section are not as good as those of the previous section. In only two of the five maps is a lower R factor observed, and in only three of the five maps is the best structure more complete

I also test whether a third iteration of ACMI might be beneficial, by measuring the mean phase error using the second iteration's calculated phases. Figure 5.12 shows the average phase error over the five maps used for this section's experiments. This plot shows that there may be some benefit to a third iteration of ACMI, as the phase error is reduced further after two iterations of ACMI (though not nearly as much as it is reduced from experimental phases to iteration 1's calculated phases).

Although ACMI consistently reduces the phase error of my ten testset maps, I am unable to realize a corresponding improvement in model quality. Further work is needed to improve ACMI's utility when used to iteratively improve poor initial experimental phasing.

5.6 Conclusions and future work

This chapter introduced ACMI-PF, an algorithm that uses particle filtering to automatically produce a set of all-atom protein models for a given electron-density map. Particle filtering considers growing an ensemble of all-atom protein models, at each iteration sampling the layout of an

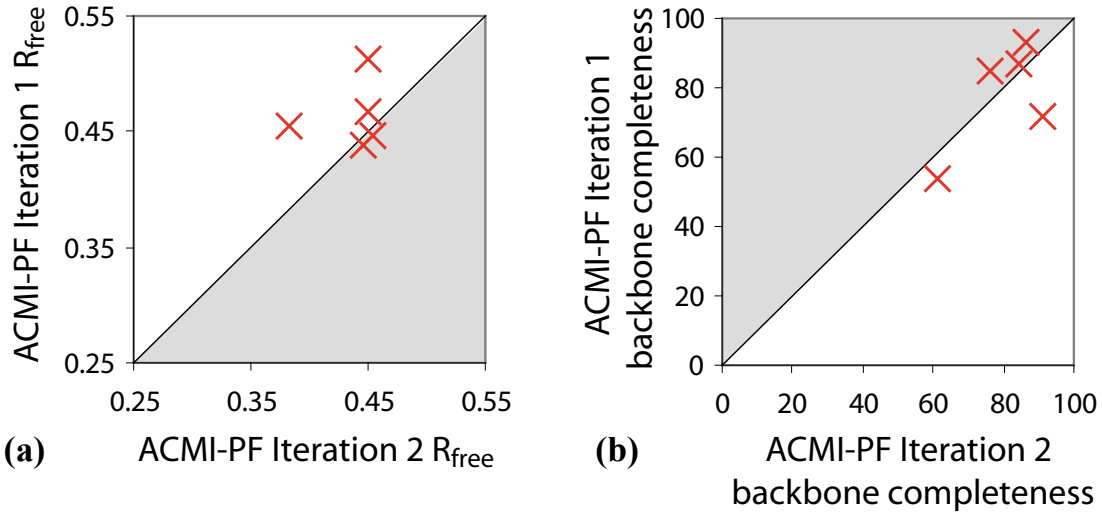


Figure 5.11 Scatterplots comparing results after one and two iterations of ACMI. I compare the two iterations' models in terms of (a) free R factor and (b) backbone completeness of PF's single best structure. Shaded regions indicate maps for which the second iteration produces a better model.

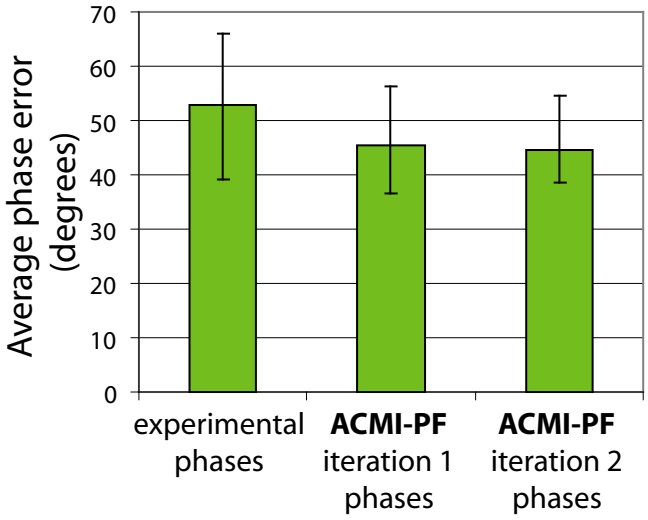


Figure 5.12 Mean phase error as a function of iteration. I compare the average phase error of experimental phases, one-iteration calculated phases, and two-iteration calculated phases. Error bars show the range of phase errors over the five maps.

additional amino acid. The method builds on the method described in the previous two chapters, where I use probabilistic inference to compute a probability distribution of each amino acid's Cα location. In addition to producing an all-atom protein model, ACMI-PF addresses shortcomings of my previous work, producing a set of *physically feasible* protein structures that best explain the density map.

My results indicate that ACMI-PF generates much more accurate and more complete models than other state-of-the-art automated interpretation methods for poor-resolution density map data. ACMI-PF produces accurate interpretations, on average finding and identifying 80% of the protein structure in poorly phased 2.5 to 3.5 Å resolution maps. Its probabilistic, model-based approach was equally good at sidechain identification, while existing algorithms usually failed to identify even 10% of the amino acids.

Using ACMI-PF, an ensemble of conformations may be easily generated using multiple runs of particle filtering. I show that sets of multiple structures generated from multiple particle filtering runs better fit the density map than a single structure. This is consistent with recent observations of the inadequacy of the single-model paradigm for modeling flexible protein molecules [11, 22, 35] and with the encouraging results of the ensemble refinement approach [74]. The ensemble description may also provide valuable information about protein conformational dynamics.

ACMI-PF's model-based approach is very flexible, and allows integration of multiple sources of "fuzzy" information, such as locations of selenium peaks which were incorporated into the calculation of the priors. In the future, it may be productive to integrate other sources of information in my model. For example, particle placement could make use of structures of distant homologues. Multiple copies of the protein in the asymmetric unit can be placed simultaneously, allowing accurate placement when the density in one copy is ambiguous. A more complicated reweighing function based on physical or statistical energy could better overcome ambiguities of unclear regions in the density map. The inclusion of these and other sources of information is possible, so long as they can be expressed in the probabilistic framework proposed here. This work could further extend the resolution (and the maximal phase error) in which automated interpretation of density maps is possible.

Chapter 6

Pictorial Structures for Atom-level Models

I present herein an alternate approach to sidechain placement in electron-density maps. Unlike the method in Chapter 5, this method is suitable for identifying individual atoms in novel sidechain conformations. The method could be used as an alternative to last chapter's ACMI-PF, placing individual atom's on ACMI-BP's maximum-marginal backbone model. However, as this approach is atom-based, it is best-suited for medium-resolution (2-3 Å resolution) maps.

I describe DEFT (DEFormable Template), an algorithm that uses pictorial structures to build a flexible all-atom protein model from a protein's amino-acid sequence. Matching this pictorial structure into the density map using Felzenszwalb and Huttenlocher's fast matching algorithm [31] is an alternate way of automating sidechain interpretation in electron-density maps. This chapter also describes several general extensions to Felzenszwalb and Huttenlocher's pictorial-structure matching algorithm. These extensions are necessary to accurately interpret density-map data. Much of the material in this chapter was previously published [26].

6.1 Pictorial structures

Pictorial structures [33] model classes of objects using a single flexible template. The template represents the object class as a collection of parts linked in a graph structure. Each edge defines a relationship between the two parts it connects. For example, a pictorial structure for a face may include the parts "left eye" and "right eye." Edges connecting these parts could enforce the constraint that – in any image containing a face – the left eye is near to the right eye. A dynamic programming (DP) matching algorithm of Felzenszwalb and Huttenlocher (hereafter referred to as the *F-H matching algorithm*) allows pictorial structures to be quickly matched into a two-dimensional image. The matching algorithm finds the globally optimal position and orientation of each part in the pictorial structure, making some simplifying assumptions concerning independence of parts and connections.

Formally, one represents a pictorial structure as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, with $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ the set of parts, and edge $e_{ij} \in \mathcal{E}$ connecting neighboring parts v_i and v_j if an explicit dependency exists between the configurations of the corresponding parts. Each part v_i is assigned a configuration l_i describing the part's position and orientation in the image. The model assumes Markov independence: the probability distribution over a part's configurations is conditionally independent of every other part's configuration, given the configuration of all the part's neighbors in the

graph. The model assigns each edge a *deformation cost* $d_{ij}(l_i, l_j)$, and each part a *"mismatch" cost* $m_i(l_i, \mathbf{I})$. These functions are the negative log likelihoods of a part (or pair of parts) taking a specified configuration, given the pictorial structure model.

The matching algorithm finds the maximum-likelihood configuration of parts in the image. That is, it finds the configuration L of parts in model T in image I maximizing:

$$\begin{aligned} P(L|\mathbf{I}, \mathbf{\Theta}) &\propto P(\mathbf{I}|L, \mathbf{\Theta}) \times P(L|\mathbf{\Theta}) \\ &\propto \exp(\sum_{v_i \in \mathcal{V}} m_i(l_i, \mathbf{I})) \times \exp(\sum_{(e_{ij}) \in \mathcal{E}} d_{ij}(l_i, l_j)) \end{aligned} \tag{6.1}$$

By monotonicity of exponentiation, the configuration maximizing this probability *minimizes*

$$\sum_{v_i \in V} m_i(l_i, \mathbf{I}) + \sum_{v_i, v_j \in E} d_{ij}(l_i, l_j) \tag{6.2}$$

The F-H matching algorithm places several additional limitations on the pictorial structure. The object's graph must be tree structured (cyclic constraints are not allowed), and the deformation-cost function must take the form $||T_{ij}(l_i) - T_{ji}(l_j)||$, where T_{ij} and T_{ji} are arbitrary functions and $|| \cdot ||$ is some norm (e.g. Euclidian distance).

6.2 Building a flexible atomic model

Given a three-dimensional map containing a large molecule and the topology (i.e., for proteins, the amino-acid sequence) of that molecule, DEFT's task is to determine the Cartesian coordinates in the 3D density map of each atom in the molecule. My idea is to build a pictorial structure corresponding to a protein, and search for it in the image.

However, because of the size of proteins, such an approach is intractable. Instead we, like others [55, 111], consider a two-phased approach, where first the protein backbone is placed, then sidechains are placed onto the backbone (potentially modifying the backbone placement). DEFT finds the coordinates of sidechain atoms simultaneously by first building a pictorial structure corresponding to a protein's amino acid, then uses F-H matching to optimally place the model into the density map. This section describes construction of a molecule's graph, and DEFT's *deformation-cost function* and *matching-cost function*.

DEFT's deformation cost is related to the probability of observing a particular configuration of a molecule. Ideally, this function is proportional to the inverse of the molecule's potential energy, since configurations with lower potential energy are more likely observed in nature. Unfortunately, such a potential function would be very complicated and cannot be approximated in a tree-structured pictorial structure, as it includes energies from both bonded and non-bonded atoms.

My solution is to *only consider the relationships between covalently bonded atoms*. DEFT constructs a pictorial structure graph where vertices correspond to non-hydrogen atoms, and edges correspond to covalent bonds between atoms. The cost function each edge defines maintain invariants such as interatomic distance and bond angles while allowing free rotation around the bond.

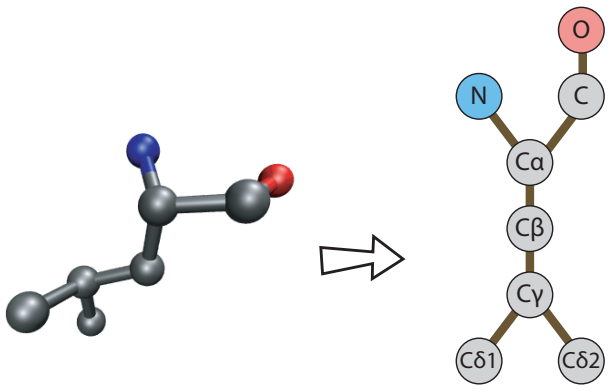


Figure 6.1 DEFT's construction of the pictorial-structure graph given an amino acid. Atoms in the molecule correspond to nodes in the graph, while edges model covalent bonds between atoms.

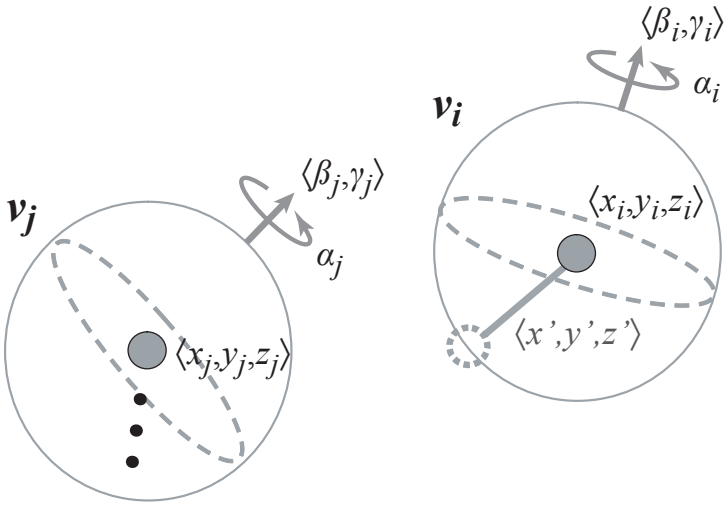


Figure 6.2 The *screw-joint*, which connects atoms in DEFT's model. In the directed version of the pictorial-structure graph, v_i is the parent of v_j. By definition, v_j is oriented such that its local z-axis is coincident with its ideal bond orientation $\mathbf{x_{ij}} = (x_{ij}, y_{ij}, z_{ij})$ in v_i. These bond parameters $\mathbf{x_{ij}}$ are learned by DEFT.

Given the protein's amino acid sequence, model construction – illustrated in Figure 6.1 – is trivial. Each part's configuration is defined by six parameters: three translational, three rotational (Euler angles α, β, and γ). For the cost function, I define a new connection type in the pictorial-structure framework, the *screw-joint*, illustrated in Figure 6.2.

The screw-joint's cost function is mathematically specified in terms of a directed graph analogous to the pictorial-structure's undirected graph. Since the graph is constrained by the fast matching algorithm to take a tree structure, I arbitrarily pick a node to serve as the root and direct every edge in the graph to point toward the root. This lets me define the screw joint in terms of a *parent atom* and a *child atom*. I have chosen to allow each object in the graph (that is, each atom) to freely rotate about its local z axis, so I want each child's z axis coincident with the axis connecting the child and the parent.

Under this constraint, the ideal geometry between child and parent is specified by *three parameters* stored at each edge, $\mathbf{x_{ij}} = (x_{ij}, y_{ij}, z_{ij})$. These three parameters define the optimal translation between parent and child, *in the coordinate system of the parent* (which in turn is defined such that its z-axis corresponds to the axis connecting it to its parent).

In constructing the deformation-cost function d_{ij}, I define the function T_{ij}, which maps a parent v_i's configuration l_i into the configuration l_j of that parent's ideal child v_j. Given parameters $\mathbf{x_{ij}}$ on the edge between v_i and v_j, the function is defined

$$T_{ij}(\langle x_i, y_i, z_i, \alpha_i, \beta_i, \gamma_i \rangle) = \langle x_j, y_j, z_j, \alpha_j, \beta_j, \gamma_j \rangle \tag{6.3}$$

where

$$\begin{aligned}
\alpha_j &= \alpha_i \\
\beta_j &= \arctan(\frac{\sqrt{(x'^2 + y'^2)}}{-z'}) \\
\gamma_j &= \frac{\pi}{2} + \arctan(\frac{y'}{x'}) \\
\begin{pmatrix} x_j \\ y_j \\ z_j \end{pmatrix} &= \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} + \begin{pmatrix} x' \\ y' \\ z' \end{pmatrix}
\end{aligned}$$

Here, $(x', y', z')^{\mathrm{T}}$ is the rotation of the bond parameters $(x_{ij}, y_{ij}, z_{ij})^{\mathrm{T}}$ to world coordinates. That is,

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \mathbf{R}_{\alpha_\mathbf{i}\beta_\mathbf{i}\gamma_\mathbf{i}} \times \begin{pmatrix} x_{ij} \\ y_{ij} \\ z_{ij} \end{pmatrix} \tag{6.4}$$

$\mathbf{R}_{\alpha_\mathbf{i}\beta_\mathbf{i}\gamma_\mathbf{i}}$ is the rotation matrix corresponding to Euler angles $(\alpha_i, \beta_i, \gamma_i)$. The expressions for β_j and γ_j define the optimal orientation of each child: $+z$ coincident with the axis that connects child and parent.

The F-H matching algorithm requires that my cost function takes a particular form, specifically, it must be some norm. The screw-joint model sets the deformation cost between parent v_i and child v_j to the distance between child configuration l_j and $T_{ij}(l_i)$, the ideal child configuration given parent configuration l_i (T_{ji} – the analogue to Equation 6.3's T_{ij} – is simply the identity function).

I use the 1-norm distance, weighted in each dimension,

$$\begin{aligned} d_{ij}(l_i, l_j) &= ||T_{ij}(l_i) - l_j||_1 \\ &= w_{ij}^{rotate} \times |\alpha_i - \alpha_j| \\ &\quad + w_{ij}^{orient} \times |(\beta_i - \beta_j) + \arctan(\frac{\sqrt{(x'^2 + y'^2)}}{-z'})| \\ &\quad + w_{ij}^{orient} \times |(\gamma_i - \gamma_j) + \arctan(\frac{y'}{x'})| \\ &\quad + w_{ij}^{translate} \times \big(|(x_i - x_j) - x'| + |(y_i - y_j) - y'| + |(z_i - z_j) - z'|\big) \end{aligned} \tag{6.5}$$

In the above equation, w_{ij}^{rotate} is the cost of rotating about a bond, w_{ij}^{orient} is the cost of rotating around any other axis, and $w_{ij}^{translate}$ is the cost of translating in Cartesian space. DEFT's screw-joint model sets w_{ij}^{rotate} to 0, and w_{ij}^{orient} and $w_{ij}^{translate}$ to 100. In this model, any part is free to rotate about a bond; DEFT gives high cost to any movement other than bond rotations.

DEFT's match-cost function implementation is based upon Cowtan's FFFEAR algorithm [17]. His algorithm cleverly uses fast Fourier transforms to quickly and efficiently calculate the mean squared distance between a weighted template of density and a region in a density map. Given a learned template and a corresponding weight function, FFFEAR uses maximum likelihood to determine the probability that the weighted template generated a region of density in the density map. The weight function is easily computed as the inverse variance of a point in the template over training data.

For each non-hydrogen atom in the protein, I create a target template corresponding to a neighborhood around that particular atom, using a training set of crystallographer-solved structures. I build a separate template for each atom type (e.g., the Cβ – the 2nd sidechain carbon – of leucine and the backbone oxygen of serine) producing 171 different templates in total. Each atom's m_i function is simply the FFFEAR-computed mismatch score of that part's template over all positions and orientations.

Once I construct the model, the parameters for the model – including the optimal orientation $\mathbf{x_{ij}}$ corresponding to each edge as well as the template for each part – are learned by training the model on a set of crystallographer-solved structures. Learning the orientation parameters is fairly simple: for each atom I define canonic coordinates (where $+z$ corresponds to the axis of rotation). For each child, I record the distance r and orientation (θ, ϕ) in the canonic coordinate frame. I average over all atoms of a given type in the training set to determine average parameters $\bar{r}$, $\bar{\theta}$, and $\bar{\phi}$. Converting these averages from spherical to Cartesian coordinates gives the ideal orientation parameters $\mathbf{x_{ij}}$.

A similarly defined canonic coordinate frame is employed when learning the model templates; in this case, DEFT's learning algorithm computes target and weight templates based on the average and inverse variance over the training set, respectively. Figure 6.3 shows an overview of the learning process. My implementation uses Cowtan's Clipper library [18].

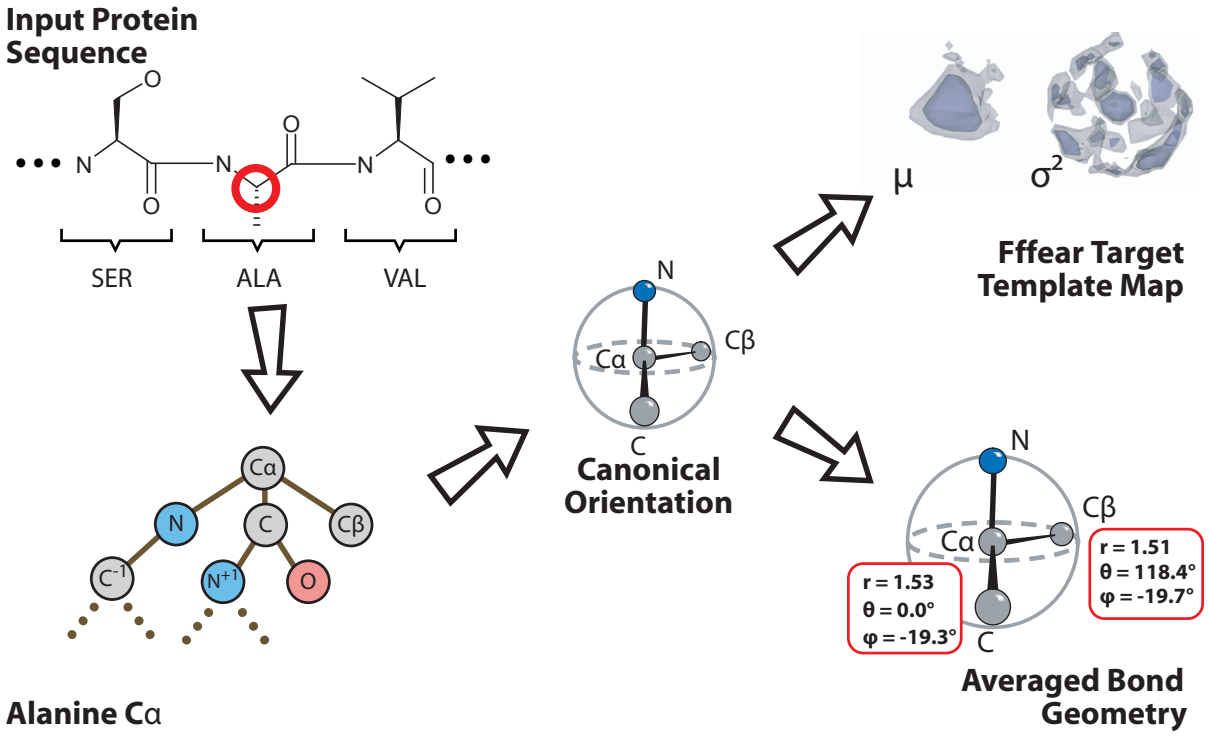


Figure 6.3 DEFT's parameter-learning process. For each atom of a given type – here alanine's Cα – DEFT rotates the atom into a canonic orientation. It then averages over every atom of that type to produce a density template and the average bond geometry.

For each part in my model, DEFT searches through a six-dimensional conformation space $(x, y, z, \alpha, \beta, \gamma)$, breaking each dimension into a number of discrete bins. The translational parameters x, y, and z are sampled on a regular grid covering the unit cell. Rotational space is (approximately) uniformly sampled using an algorithm described by Mitchell [80].

6.3 Model enhancements

Upon initial testing, pictorial-structure matching performed rather poorly at the density map interpretation task. Consequently, I added two routines – a collision-detection routine, and an improved template-matching routine – to DEFT's pictorial-structure matching. Both enhancements can be applied to pictorial-structure matching in general, and are not specific to electron-density map interpretation.

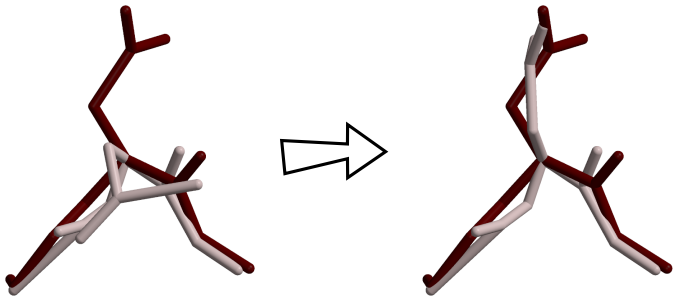


Figure 6.4 An example of DEFT's collision correction. (a) A model with a "collision" (the predicted molecule is in the lighter color), where the amino acid's sidechain is placed coincident with the backbone. (b) DEFT perturbs the structure, and finds the correct sidechain conformation.

6.3.1 Collision detection

Closer investigation reveals much of the algorithm's poor performance is due to distant chains colliding. Since DEFT only models covalent bonds, the matching algorithm sometimes returns a structure with non-bonded atoms impossibly close together. These collisions clearly were a problem in DEFT's initial implementation. Figure 6.4 shows one such collision corrected by the algorithm.

Given a candidate solution, it is straightforward to test for spatial collisions: I simply test if any two atoms in the structure are impossibly (physically) close together. If a collision occurs in a candidate, DEFT perturbs the structure. Though the globally optimal match is no longer returned, this approach works well in practice. If two atoms are both aligned to the same space in the most probable conformation, it seems quite likely that *one* of the atoms belongs there. Thus, DEFT handles collisions by assuming that at least one of the two colliding branches is correct. When a collision takes place, DEFT finds the closest branch point above the colliding nodes. That is, DEFT finds the root y of the smallest subtree containing all colliding nodes. DEFT iterates through each child x_i of this root, matching the subtree rooted at x_i, while keeping the remainder of the tree fixed. The change in score for each perturbed branch is recorded, and the one resulting in the smallest score increase is the one DEFT keeps.

Algorithm 6.1 describes the collision-avoidance algorithm. In the case that the colliding node is due to a chain wrapping around on itself (and not two branches running into one another), the root y is defined as the colliding node nearest to the top of the tree. Everything below y is matched anew while the remainder of the structure is fixed.

6.3.2 Improved template matching

In my original implementation, DEFT learned a template by averaging over each of the 171 atom types. For example, for each of the 12 (non-hydrogen) atoms in the amino-acid tyrosine I

Algorithm 6.1 DEFT's collision-handing routine.

input: An illegal pictorial-structure configuration $L = \{l_1, l_2, \ldots, l_n\}$
output: A legal perturbation L'

```
// Repeat this process until all collisions are resolved
```
while *configuration L contains one or more collisions* **do**
```
    // Find root of smallest subtree containing all colliding nodes
    // In each iteration, this root is guaranteed to increase in depth,
    //     thus the algorithm will terminate
```
    $X \leftarrow$ all nodes in L illegally close to some other node
    $y \leftarrow$ root of smallest subtree containing all nodes in X
```
    // Now perturb each subtree of y, holding remainder of tree fixed
```
    foreach *child x_i of y* **do**
        $L_i \leftarrow$ optimal position of subtree rooted at x_i fixing remainder of tree
        $score_i \leftarrow \mathbf{score}(L_i)\mathbf{score}(\text{subtree of } L \text{ rooted at } x_i)$
    end
```
    // Choose the perturbation with the minimum cost
```
    $i_{min} \leftarrow \arg\min_i score_i$
    $L' \leftarrow$ replace subtree rooted at x_i in L with $L_{i_{min}}$
    $L \leftarrow L'$
end
```
// Return the perturbed structure
```
return L

build a single template – producing 12 tyrosine templates in total. Not only is this inefficient – giving DEFT numerous sets of redundant templates to match against the unsolved density map – but for some atoms in flexible sidechains, averaging blurs features more than a bond away, losing information about an atom's neighborhood.

DEFT improves the template-matching algorithm by modeling templates using a mixture of Gaussians, a generative model where each template is modeled using a mixture of basis templates. That is, given a set of N basis templates $\mu_i(\vec{x})$, $i = 1, \ldots, N$, each with variance $\sigma_i^2(\vec{x})$, we model the density around each atom, $\rho(\vec{x})$ using a linear combination of basis templates:

$$\rho(\vec{x}) = \sum_{i=1}^{N} h_i \cdot \rho(\vec{x}) \tag{6.6}$$

Each μ_i and σ_i^2 is iterative learning using expectation maximization (EM) [21]. In each iteration of the algorithm I first compute the *a priori* likelihood of each template being generated by a particular cluster mean (the E step). Then I use these probabilities to update the mu_i's and $sigma_i$'s. After convergence, I use the each cluster mean (and variance) as an FFFEAR search target. DEFT uses this algorithm to build $N = 24$ basis templates for matching.

6.4 Experimental studies

I test DEFT on a set of four solved experimentally-phased density maps provided by the Center for Eukaryotic Structural Genomics at the University of Wisconsin–Madison. Details of the dataset are shown in Section A.1, in Appendix A. To test my algorithm with poor-quality data, I downsampled each of the maps to 2.5, 3 and 4Å by removing higher-resolution reflections and recomputing the density.

My experiments are conducted under the simplifying assumption that the backbone Cα's of the protein were known to within some error factor. This assumption is fair; alternate approaches exist for backbone tracing in density maps [40]. DEFT simply walks along the backbone, placing atoms one residue at a time.

I split the dataset into a training set of about 1000 residues and a test set of about 100 residues (from a protein not in the training set). Using the training set, I built a set of templates for matching using FFFEAR. The templates extended to a 6Å radius around each amino-acid's Cα on a grid with 0.5Å sampling. Two sets of templates were built and subsequently matched: a large set of 171 produced by averaging all training set templates for each atom type, and a smaller set of 24 learned through by the EM algorithm. I ran DEFT's pictorial-structure matching algorithm using both sets of templates, with and without collision-detection code.

Although placing individual atoms into the sidechain is fairly quick, taking less than six hours for a 200-residue protein, precomputing match scores using FFFEAR is very CPU-demanding. For each of the 171 templates, FFFEAR takes 3-5 CPU-hours to compute the match score at each location in the image, for a total of one CPU-month to match templates into each protein. Fortunately the task is trivially parallelized; I regularly do computations on over 100 computers simultaneously using the University of Wisconsin's Condor distributed computing environment [112].

The results of all tests are summarized in Figure 6.5. Using individual-atom templates and the collision detection code, the all-atom RMS deviation varied from 1.38Å at 2Å resolution to 1.84Å at 4Å resolution. Using the EM-based clusters as templates produced slight or no improvement. However, much less work is required; only 24 templates need to be matched to the image instead of 171 individual-atom templates. Finally, it was promising that collision detection leads to significant error reduction.

It is interesting to note that individually using the improved templates and using the collision avoidance both improved the search results; however, using both together the results were a bit worse than with collision detection alone. More research is needed to get a synergy between the two enhancements. More work is also needed balancing between the number of templates and template size. The match-cost function is a critically important part of DEFT and improvements there will have the most profound impact on the overall error.

6.5 Conclusions and future work

My DEFT algorithm applies the F-H pictorial-structure matching algorithm to the task of interpreting electron-density maps. In the process, I extended the F-H algorithm in three key ways. In order to model atoms rotating in 3D, I designed another joint type: the screw joint. I also

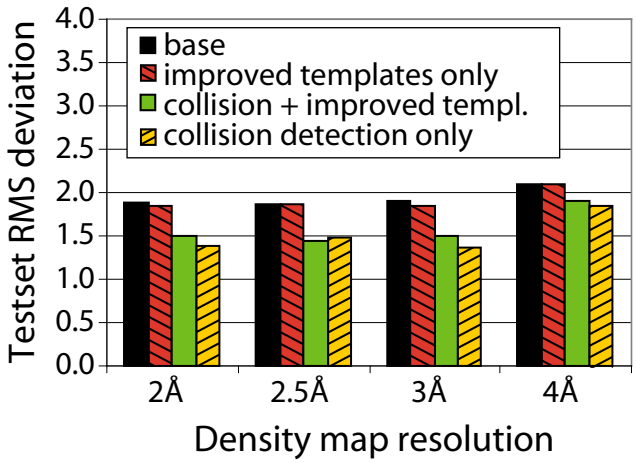


Figure 6.5 DEFT's leave-one-protein-out testset errors at four resolutions, under four strategies.

developed extensions to deal with spatial collisions of parts in the pictorial-structure model, and implemented a slightly better template-construction routine. These enhancements can be applied to pictorial-structure matching in general, and are not specific to the task presented here (protein sidechain placement).

DEFT bridges the gap between two model-fitting approaches for interpreting electron-density maps. A number of techniques [53, 75, 94] have been shown to do a good job of placing individual atoms into a density map, but all have a ceiling resolution of about 2.5 to 3Å. On the other hand, FFFEAR has had success finding secondary-structure elements in very poor resolution maps, but is unable to place individual atoms or even residues not contained within protein secondary structures. My work attempts to extend the resolution threshold at which individual atoms can be identified in electron-density maps. Doing so allows for more rapid and less expensive collection of protein structures. DEFT's flexible template matching combines weakly-matching image templates to extract atomic coordinates from maps where individual atoms have been blurred away. No other approach has investigated sidechain refinement in structures of this poor resolution.

A different optimization algorithm that handles cycles in the pictorial-structure graph would better handle collisions (allowing edges between non-bonded atoms). In recent work [107], loopy belief propagation [66] has been used with some success (though with no optimality guarantee). I plan to explore the use of belief propagation in pictorial-structure matching, adding edges in the graph to avoid collisions.

Another direction for future work is incorporating DEFT's sidechain identification and placement into ACMI. Instead of using a set of rigid 5-mer templates to locate individual amino acids, one could conceivably use DEFT as a vertex potential (that is, the "amino-acid detector.") An advantage of DEFT – and corresponding weakness of ACMI – is that DEFT's flexible sidechain model is able to handle proteins with novel or rare sidechain conformations.

Chapter 7

Improving the Efficiency of Belief Propagation

Chapter 3 presented a protein Markov-field model and a probabilistic inference algorithm – ACMI-BP – specialized for this model. That algorithm is able to infer the marginal distribution of each amino acid's location in a density map. This chapter generalizes ACMI-BP's model and inference algorithm.

I develop a part-based object-recognition framework, suitable for mining complex 3D objects in detailed 3D images. As with the pictorial-structure matching of the previous chapter, my framework models objects as a collection of connected parts. The algorithm's key component is an efficient inference algorithm, based on belief propagation [93], that finds the optimal layout of parts, given some input image. Belief propagation (BP) is well suited to this task. However, for large objects with many parts, even BP may be intractable.

Consequently, I introduce AGGBP, a message aggregation scheme for BP, designed for inferring the layout of complex objects. In AGGBP, groups of BP messages are approximated as a single AGGBP message. This gives AGGBP a message update analogous to that of mean-field methods [61, 95]. For objects consisting of N parts, the approximation reduces CPU time and memory requirements from O(N^2) to O(N). AGGBP is applicable to any model expressible in my part-based framework (with some accuracy tradeoffs).

Finally, I apply AGGBP to both real-world and synthetic tasks. First, I use my framework to recognize protein fragments in three-dimensional images. Scaling BP to this task for even average-sized proteins is infeasible without my enhancements. I then use a synthetic "object generator" to test my algorithm's ability to locate a wide variety of part-based objects. Earlier versions of this chapter were previously published [24, 25].

7.1 Introduction

Several publications – including the pictorial-structure matching of the previous chapter – have explored the use of part-based models for recognizing generic objects in images [31,56,106]. These models represent physical objects as a graph: a collection of rigid parts (vertices in the graph) connected by flexible joints (edges in the graph). Potential functions associated with vertices describe preferred locations for each part in the image, while edge potential functions describe preferred conformations between pairs of parts. An inference algorithm determines the most probable location of each part in the model given the image and the conformational constraints on the object.

However, previous work has only considered simple objects with relatively few parts, (and often using two-dimensional image data). I present a part-based object-recognition algorithm specialized to objects with hundreds of parts in detailed, three-dimensional images.

To effectively identify complex 3D objects, I introduce an efficient message-passing inference algorithm based on *belief propagation* [93]. Message-passing algorithms like belief propagation are an efficient yet powerful technique for inference in graphical models. Belief propagation (BP) is a message-passing method for exactly computing marginal distributions in tree-structured graphs. In graphs with arbitrary topologies, no such optimality is guaranteed. Empirically, however, "loopy BP" often provides accurate approximations, when exact inference methods are intractable [34, 77, 116].

For very large, highly-connected graphs, with large input images, even loopy BP may not offer enough efficiency. In near-fully connected graphs, with hundreds or thousands of vertices, approximations to BP's messages may be necessary to compute approximate marginal distributions in a reasonable amount of time. I describe AGGBP (for *aggregate BP*), which approximates groups of BP messages with a single message. This single composite message turns out to be quite similar to the message update for mean-field methods. I illustrate that, for densely connected graphs with a certain type of edge potential, AGGBP reduces running time in a graph with N nodes from O(N^2) to O(N).

Additionally, I provide a method for dealing with continuously-valued variables that is efficient and does not require accurate initialization. Sudderth *et al.* [105] have developed an extension to BP, nonparametric belief propagation (NBP). NBP represents variables that have continuous non-Gaussian distributions as a mixture of Gaussians. They have also developed efficient algorithms for computing messages and taking the product of messages using these distributions. This chapter introduces an efficient variant, which alternately represents probability distributions over a continuous three-dimensional space as a set of Fourier-series coefficients. Coupled with AGGBP, this representation allows for efficient message passing and computation of message products.

Finally, I test these approximation techniques using both real-world and synthetic data. The first testbed is a variant of the density map interpretation described throughout this thesis, and involves searching for short protein fragments in a masked portion of an electron-density map. The second testbed uses a synthetic-object generator to test AGGBP's performance locating a wide variety of objects with various part topologies.

7.2 Modeling 3D objects

Following others [31, 33], I describe a class of objects using an undirected graphical model. Recall from Section 2.1.4 that undirected graphical models represent the joint probability distribution over a set of variables as a function defined on an undirected graph. A *pairwise undirected graphical model* (or *pairwise Markov field*) represents this joint distribution as a product of potential functions defined on each *edge* and *vertex* in the graph. That is, given variables x_s, evidence $\mathbf{y}$, vertex potentials $\psi_s(x_s|\mathbf{y})$ and edge potentials $\psi_{st}(x_s, x_t)$, the probability of some setting of the

x_s's is given:

$$p(\mathbf{x}|\mathbf{y}) \propto \prod_{(s,t)\in\mathcal{E}} \psi_{st}(x_s, x_t) \times \prod_{s\in\mathcal{V}} \psi_s(x_s|\mathbf{y}) \tag{7.1}$$

To represent a 3D object in such a model, I construct a graph where vertices correspond to parts in the object, while edges correspond to constraints between pairs of parts. Each x_s describes the 3D position of part s (in addition, x_s may contain information about the orientation of part s or other internal parameters).

Then, to describe an object using a graphical model, I need three pieces of information: a *part graph*, each node's *observation potential*, and each edge's *structural potential*. Given a graph describing an object, these potential functions may be learned from a set of previously solved problem instances.

In my 3D object recognition framework, objects have a fully connected *part graph*. Most edges are associated with identical, weak (diffuse) potentials, ensuring that no pair of parts may occupy the same 3D space. However, a sparse subset of the graph (the "skeleton") connects highly correlated variables. As an illustration, consider using a graphical model for recognizing people in images, as in Figure 7.1. In this model, a sparsely connected skeleton connects highly correlated nodes. For example, the head and body are connected in this skeletal structure, because the position of the head and the position of the body are highly correlated.

However, many other pairs of nodes – such as the left leg and the left arm – are not connected in the skeletal structure, yet their labels are not completely (conditionally) independent. There is a weak dependency (beyond the dependency passed through the position of the body): the two parts may not occupy the same location in 3D space. As this constraint is not implicitly modeled by the chain that connects them in the skeletal structure, an edge between them enforcing this constraint is necessary. My model refers to these edges as *occupancy edges*. For example, when modeling a hand [106], occupancy edges are required to ensure two fingers do not occupy the same space. The potential associated with these edges is typically very diffuse; it is non-zero everywhere except in a small neighborhood around the origin (in each part's local coordinates).

Each part's observation potential is usually based on the application of a simple classifier (i.e., a "part detector"). At each location in 3D space, it returns the probability that a particular part is at that location. Individual part potential functions may use template matching (as in Chapters 3 and 4), color matching [31], edge detection [105], or any other method. One strength of the part-based framework is that individual observation potentials need not be particularly accurate, as belief propagation is able to infer the true location using the combined power of many weak detectors [31].

As illustrated in the person-detector example, structural potentials are broken into two types: *skeletal potentials* (or *adjacency potentials* for a linear skeleton) model the relationship between parts connected through an object's skeleton, while *occupancy potentials* model the relationship between all other pairs of nodes. Skeletal potentials may take an arbitrary form, learned from a set of allowable object conformations. They may be a function of position as well as orientation of the 3D object, making use of additional latent variables. Occupancy potentials take the form of a step function (using a "hard collision" model) or a sigmoidal function (using a "soft collision" model), only taking a nonzero value if two parts are sufficiently far apart. In my model, occupancy

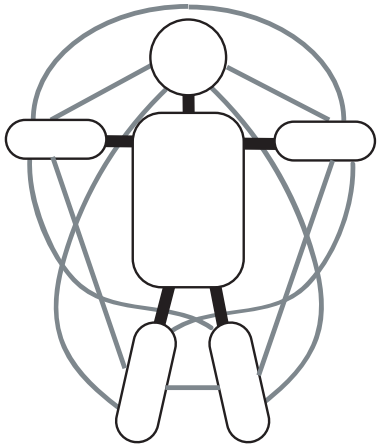


Figure 7.1 A graphical model for recognizing a person in an image. Thicker dark edges illustrate the highly-correlated "skeleton" of the model, while thinner light edges are weakly-correlated occupancy edges, which ensure two parts do not occupy the same 3D space.

potentials only depend on the position of the connected parts, treating individual parts as spheres (this constraint is necessary for efficient inference).

7.3 Scaling belief propagation

Given an image and some object's graphical model, *inference* finds the most-probable location of each of the object's parts in the image. Because the object graph is fully connected, with many loops, exact inference methods either will not work (e.g., tree-based methods) or are intractable (e.g., exhaustive methods). Instead, one is forced to rely on approximate inference methods, like those described in Section 2.1.4.1. My object-recognition framework uses a variant of loopy belief propagation.

Belief propagation – described in Section 2.1.4.2 – was originally intended for small, sparsely connected graphs. In large, highly-connected graphs, the number of messages quickly becomes overwhelming. To make BP tractable in these types of graphs, I designed and developed a BP variant, AGGBP, which approximates some subset of outgoing messages at a single node with a single message, significantly reducing the number of messages computed an stored in each iteration of the algorithm.

7.3.1 BP message aggregation

In the undirected graphical models used for 3D object recognition, pairs of nodes along *skeleton* edges are highly correlated. Consequently, messages along these edges have a high information

content. It is important to exactly compute messages along these edges. Coarse approximations – like those used in mean-field methods [61] – introduce too much error.

However, in these graphs, the *majority* of edges are *occupancy* edges, which enforce the constraint that two parts cannot occupy the same 3D space. The potential functions associated with these edges are *weak* – that is, when convoluted with a node's belief, they tend to spread the probability mass a lot – and messages along these edges carry little (but still some!) information. It is along these edges that I make some approximations.

Formally, BP's message update, given by Equation 2.9, can be alternately written as (again, the explicit dependence of the message on $\mathbf{y}$ is dropped for clarity):

$$m^n_{t\to s}(x_s) \leftarrow \alpha_1 \int_{x_t} \psi_{st}(x_s, x_t) \times \frac{\hat{b}^n_t(x_t|\mathbf{y})}{m^{n-1}_{s\to t}(x_t)} dx_t \tag{7.2}$$

The denominator in the above, $m^{n-1}_{s\to t}(x_t)$ is a term that serves to avoid double-counting or "feedback," making the method exact in tree-structured graphs. In loopy graphs, such feedback – through the graph's loops – in unavoidable. For messages along occupancy edges this denominator – the previously received occupancy message – carries little information. Since I am convoluting this already-weak denominator by the occupancy potential *again*, its contribution to the final message is negligible. Thus, AGGBP drops it with little loss of accuracy.

This gives an update equation more like the naïve mean-field update:

$$m^n_{t\to s}(x_s) \leftarrow \alpha_2 \int_{x_t} \psi_{st}(x_s, x_t) \times \hat{b}^n_t(x_t|\mathbf{y})\, dx_t \tag{7.3}$$

The key advantage of doing this – assuming that the structural potential ψ_{st} is identical along all occupancy edges – is *all occupancy messages outgoing from a single node are identical*. For the remainder of this section, I will refer to these approximate messages as $m_{t\to *}(x_*)$.

Assuming identical ψ_{st}'s, AGGBP reduces the number of occupancy messages computed from O(N^2) to O(N) in a model with N parts. However, *updating* the belief for some part still requires multiplying all the incoming occupancy messages times all the incoming skeletal messages; for an N-part model, this is still potentially $O(N^2)$. To reduce this complexity, I utilize the fact that each node receives this broadcast message from all but a few nodes in the graph: its neighbors (in the skeleton graph) and itself. I instead send all these aggregate messages to a central accumulator:

$$ACC(x_*) \leftarrow \prod_{t=1}^{N} m_{t\to *}(x_*) \tag{7.4}$$

I use the accumulator to efficiently update a node's belief, by sending – in a single message – the product of all occupancy messages.

Figure 7.2 illustrates AGGBP when the graph is a chain (generalizing this to arbitrary topologies is straightforward). For a chain, one computes the product of incoming messages, using this

accumulator, as:

$$\hat{b}_1 \leftarrow \psi_1 \times ACC \times \frac{m_{2\rightarrow 1}}{m_{1\rightarrow *} \times m_{2\rightarrow *}}$$
$$\hat{b}_2 \leftarrow \psi_2 \times ACC \times \frac{m_{1\rightarrow 2} \times m_{3\rightarrow 2}}{m_{1\rightarrow *} \times m_{2\rightarrow *} \times m_{3\rightarrow *}}$$
$$\vdots$$
$$\hat{b}_k \leftarrow \psi_k \times ACC \times \frac{m_{k-1\rightarrow k} \times m_{k+1\rightarrow k}}{m_{k-1\rightarrow *} \times m_{k\rightarrow *} \times m_{k+1\rightarrow *}}$$
$$\vdots$$

The numerators of these message updates contain skeletal messages, while the denominators contain the approximated occupancy messages. AGGBP reduces the runtime and memory requirements from O(N^2) to O(N) in a model with N parts. The storage benefit is especially appealing when the 3D space for each part is large, and storing O(N^2) messages is space-prohibitive. Section 7.4.1 provides a closer look at one such application where this is the case.

Algorithm 7.1 gives a pseudocode overview of AGGBP (the function arguments x_s and x_t have been dropped for clarity), with comments highlighting the differences between AGGBP and standard loopy BP (Algorithm 2.1). As I progress from node to node, instead of computing every outgoing message from each node, I compute a single composite message. The key difference between Algorithms 2.1 and 7.1 is in the inner loop, "*if s is a skeleton neighbor of t.*" For part-based object recognition, where the skeleton graph is sparsely connected, this loop is seldom entered, requiring few message calculations.

Finally, when different occupancy edges have a *different* potential functions (i.e., when parts in the model are of a different size), then my object-recognition framework may still take advantage of AGGBP, with additional approximation error. In this case, AGGBP simply computes the broadcast message from a part t using the *average* potential function outgoing from t:

$$m^n_{t\rightarrow *}(x_*) \leftarrow \alpha \int_{x_*} \frac{\sum_{u=1}^{N} \psi_{tu}(x_t, x_*)}{N} \times \hat{b}^n_t(x_t) dx_t \tag{7.5}$$

One can think of this as each part sending the *average* of all outgoing occupancy messages to every part. Section 7.4.2 explores how well AGGBP handles varying occupancy potentials.

7.3.2 Message representation

Section 2.1.4.2 describes an approach to belief propagation with continuous-valued labels based on particle filtering. However, there may be cases where these models are insufficient. In general, when using particle-based BP, reasonably accurate initialization of the Gaussian centers representing the probability distribution is necessary for accurate inference [106]. In this section, I describe an alternative belief representation that uses a Fourier-series probability density estimate [101] to represent probabilities and messages. While particle-based methods tend to concentrate on high-probability space, my approach accurately represents the probability distribution

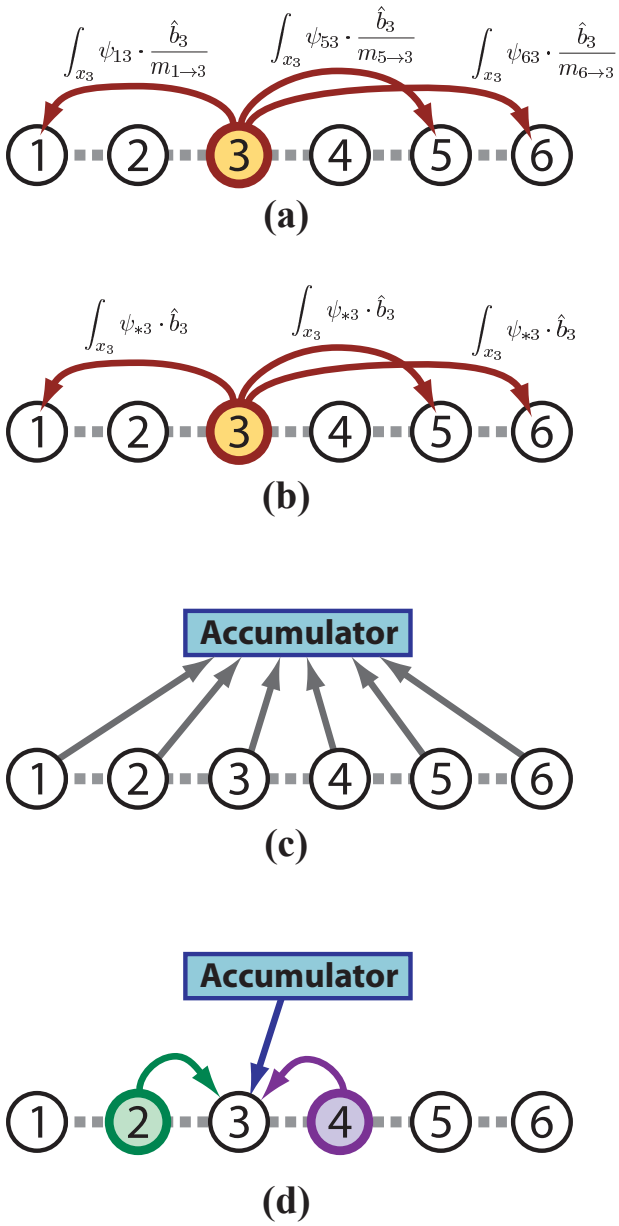


Figure 7.2 AGGBP's message aggregation approximates (a) all the outgoing messages at node 3, with (b) a single message sent to all non-adjacent nodes. Storing (c) the product of these aggregate messages in a central accumulator allows us to (d) quickly update a node's (here, node 3's) belief as the accumulator product times incoming skeletal messages, divided by the occupancy messages the node should not have received.

Algorithm 7.1 Aggregate belief propagation (AGGBP).

input: Observational potentials $\psi_s(x_s|\mathbf{y})$ and structural potentials $\psi_{st}(x_s, x_y)$

output: An approximation to the marginal $\hat{b}_s(x_s|\mathbf{y}) \approx \sum_{x_1} \cdots \sum_{x_{s-1}} \sum_{x_{s+1}} \cdots \sum_{x_N} P(\mathbf{x}|\mathbf{y})$

initialize accumulator ACC, messages m to 1
while $\hat{b}$*'s have not converged* **do**
  foreach *part* $s = 1 \ldots N$ **do**
    // Factor s's occupancy message out of the accumulator product
    $ACC \leftarrow ACC / m_{s \to *}^{n-1}$
    // Set s's belief to vertex potential *times accumulator product*
    $\hat{b}_s(x_s|\mathbf{y}) \leftarrow \psi_s \times ACC$
    foreach *part* $t = 1 \ldots N$ **do**
      // The key difference from Algorithm 2.1!
      // I only compute exact messages along the skeleton graph
      // (the accumulator handles non-skeleton edges)
      if *s is* ***skeleton*** *neighbor of t* **then**
        if $\hat{b}_t$ *has been updated* **then**
          $m_{t \to s}^n(x_s) \leftarrow \int_{x_t} \psi_{st} \times \frac{\hat{b}_t^n}{m_{s \to t}^{n-1}} dx_t$
        end
        // Multiply s's belief by the incoming message from t
        // and *divide* by t's occupancy message
        // Node s should *not* get t's occupancy message
        // (but did, from the accumulator)
        $\hat{b}_s(x_s|\mathbf{y}) \leftarrow \hat{b}_s(x_s|\mathbf{y}) \times (m_{t \to s}^n / m_{t \to *}^n)$
      end
    end
    // Compute composite occupancy message
    $m_{s \to *}^n(x_s) \leftarrow \int_{x_t} \psi_{*s} \times \hat{b}_s^n(x_s|\mathbf{y})$
    // Update accumulator product with s's contribution
    $ACC \leftarrow ACC \times m_{s \to *}^n$
  end
end

over the entire space of each random variable. Efficient message passing and message computation make this representation ideal for large, highly connected graphs.

Formally, I represent marginal distributions $\hat{b}_s^n$ as a set of three-dimensional Fourier coefficients f_k, where, given high-frequency limit K

$$\hat{b}_s^n(x_s|\mathbf{y}) \approx \sum_{k=0}^{K} f_k \times e^{-2\pi i(x_s \cdot k)} \tag{7.6}$$

Messages are represented using the same probability density estimate.

7.3.2.1 Message computation

Recall from Equation (2.8) that computing $m_{t\to s}$ requires integrating the product of edge potential $\psi_{st}(x_s, x_t)$, observation potential $\psi_s(x_s, \mathbf{y})$, and incoming message product $\prod m_{s\to t}(x_t)$ over all x_t. While this computation is difficult in general for Fourier-based density estimates, if an edge potential can be represented as a function of the *difference* between the labels of the two connected nodes, that is, $\psi_{st}(x_s, x_t) = f(||x_s - x_t||)$, then $m_{t\to s}$ is just a convolution:

$$m_{t\to s}^n(x_s) = \big(\psi_{st} * \prod m_{s\to t}^{n-1}\big)(x_s) \tag{7.7}$$

This is easily computed as the product of Fourier coefficients:

$$\mathcal{F}\big[m_{t\to s}^n(x_s)\big] = \mathcal{F}\big[\psi_{st}(x_s, x_t)\big] \times \mathcal{F}\Big[\big(\prod m_{s\to t}^{n-1}(x_t)\big)\Big]. \tag{7.8}$$

In my object-recognition framework, all occupancy potentials are represented in this manner. That is, the potential here only depends upon the *difference* between labels.

This computational shortcut was originally proposed by Felzenswalb for belief propagation in low-level vision [32]. Computing these message products is efficient, with running time $\mathrm{O}(K^3)$, where K is the high-frequency limit of the density estimate.

7.3.2.2 Message products

As shown in Equation (2.9), computing the current belief $\hat{b}_s^n$ at a given node requires taking the product of all incoming messages $m_{t\to s}(x_s)$, and multiplying it by the observation potential $\psi_s(x_s, \mathbf{y})$. Given the Fourier coefficients of all messages, I compute this multiplication in real space:

$$\hat{b}_s^n(x_s|\mathbf{y}) = \psi_s(x_s, \mathbf{y}) \times \prod \mathcal{F}^{-1}\Big[\mathcal{F}\big[m_{t\to s}^n(x_s)\big]\Big] \tag{7.9}$$

As with the message convolution, this operation is fairly efficient. Each transform and inverse transform runs in time $\mathrm{O}(K^3 \log K)$. In general, the bandwidth limit K corresponds to the grid size of the image X ($K = X$ allows me to represent the signal exactly at each grid point). Thus, assuming a (three-dimensional) $X \times X \times X$ image, computing an occupancy message takes $O(X^3 \log X)$ running time.

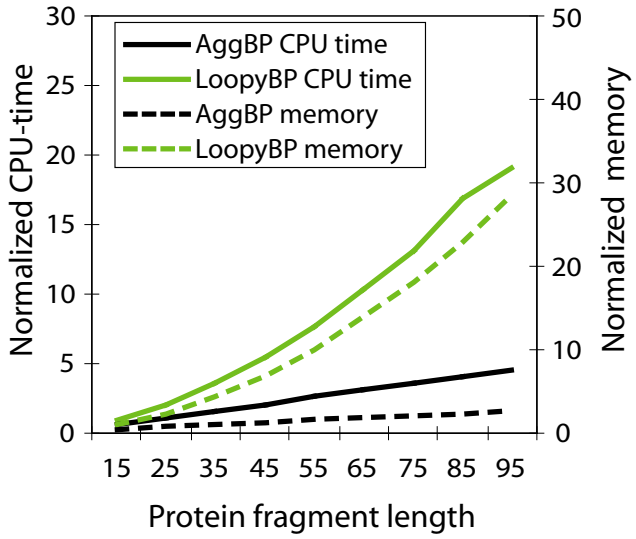


Figure 7.3 A comparison of memory and CPU time usage between AGGBP and LOOPYBP.

7.4 Experiments

In this section, I compare the standard loopy belief propagation algorithm with AGGBP, using both real-world and synthetic datasets. The real-world task is based upon locating protein fragments in 3D images; these objects consist of a chain of "parts" (amino acids). The synthetic dataset allows me to look at my algorithm's performance recognizing objects containing more complex part topologies.

7.4.1 Protein-fragment identification

This section details some experiments locating protein fragments in electron-density maps. My data set for testing (as in Chapters 3 and 4) comes from a set of ten model-phased electron-density maps from the Center for Eukaryotic Structural Genomics. I downsampled these maps to 3Å resolution by *smoothly* diminishing the intensities of higher-resolution reflections. Details of the dataset and the truncation method are given in Section A.2.

I compare standard loopy BP inference (LOOPYBP) to AGGBP on this Markov field model. LOOPYBP is unable to scale to the entire protein (as large as 500 amino acids in the testset), so to compare these two methods I consider protein *fragments* of between 15 and 65 amino acids. CPU time per iteration and memory usage of the two techniques are illustrated in Figure 7.3. Because the actual running-time and memory usage is dependant upon the size of the density map, I normalize these values, so that LOOPYBP's time and memory usage at 15 amino-acids is 1.0 (in an average-sized protein, these values are 200 MB and 120 sec, respectively). I increase fragment length until the algorithm used up my machine's 6 GB RAM and began paging; Figure 7.3 does

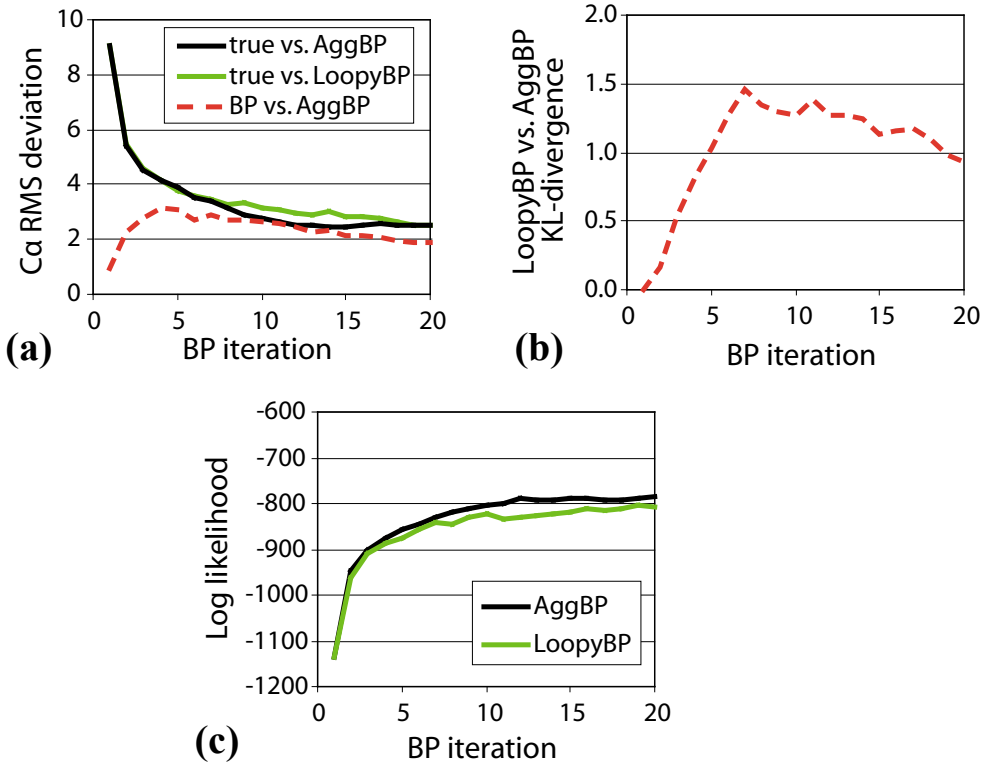


Figure 7.4 A comparison of AGGBP to loopy BP at each iteration of message passing, using (a) RMS deviation, (b) average KL-divergence of the predicted marginals, and (c) log likelihood of the maximum-marginal interpretation.

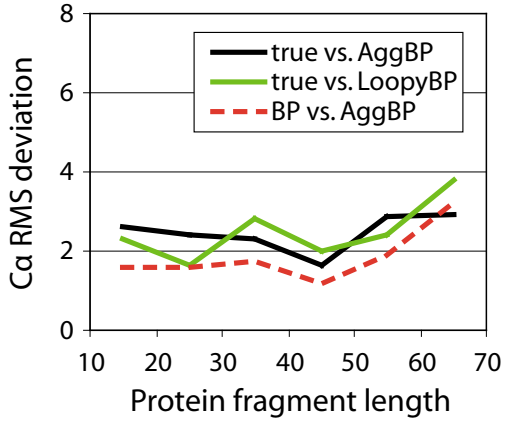


Figure 7.5 RMS error of AGGBP and LOOPYBP as a function of protein fragment size (at iteration 20).

not include time swapping to disk; however, for fragments larger than I tested, this may result in a significant performance penalty.

At each of six fragment lengths, I search for 15 different fragments of that length from 5 different proteins (in 5 different electron-density maps), for a total of 90 different target fragments. Fragments were chosen that roughly corresponded with the beginning, middle, and end of each protein chain. I ran LOOPYBP and AGGBP until convergence or 20 iterations (where one iteration is a single forward or backward pass through the protein). In each map, before searching for a fragment, I reduced the electron-density map to a neighborhood around the fragment; that is, I eliminated the electron density corresponding to portions of the protein not in the 15 to 65 amino-acid fragment. Eliminating these portions of the map provides a more-realistic model of searching for a complete protein.

Results from this experiment as a function of BP iteration appear in Figures 7.4. Panel a shows the RMS error of both AGGBP's and LOOPYBP's maximum-marginal backbone trace, in addition to the RMS deviation between the two backbone traces. Panel b shows the Kullback-Leibler (KL) divergence between the AGGBP's and LOOPYBP's estimated marginal distributions. Finally, Panel c shows the log likelihood of both AGGBP's and LOOPYBP's max-mum-marginal backbone trace. In each of these plots, the x-axis shows the number of BP iterations completed; the y-axis shows the respective performance metric. As these plots show, the solutions found by these two methods differ somewhat, however, in terms of error versus the true trace, both produce equally accurate traces. Interestingly, Figure 7.4c shows that AGGBP is finding a solution with higher log likelihood (higher is better); perhaps due to message aggregation's role as a regularizer.

Alternately, Figure 7.5 shows the RMS error as a function of protein-fragment length. This figure shows both methods seem to perform slightly worse when searching for longer fragments; still, the predicted structures are fairly accurate – considering the quality of the maps – with an average RMS error of under 4Å.

Finally, a scatterplot of log likelihoods, where each of the 90 fragments is represented as a point, is illustrated in Figure 7.6. In this figure, points below the diagonal correspond to fragments on which AGGBP produces the higher-likelihood (i.e., better) interpretation. For almost every fragment, AGGBP produces a solution with a greater log likelihood than does standard BP. This difference is statistically significant; a two-tailed, paired *t* test gives a *p* value of 0.014.

7.4.2 Synthetic-object recognition

While the protein-fragment identification testbed shows the CPU and memory savings achievable by my algorithm, it involves a rather limited part topology: the skeletal structure is just a linear chain, and each part is a constant distance apart. In this section, I construct a synthetic-object generator, that builds "part graphs" with varying branching factors, object sizes, and object "softness." I also explore approximation performance under various part-finder accuracies.

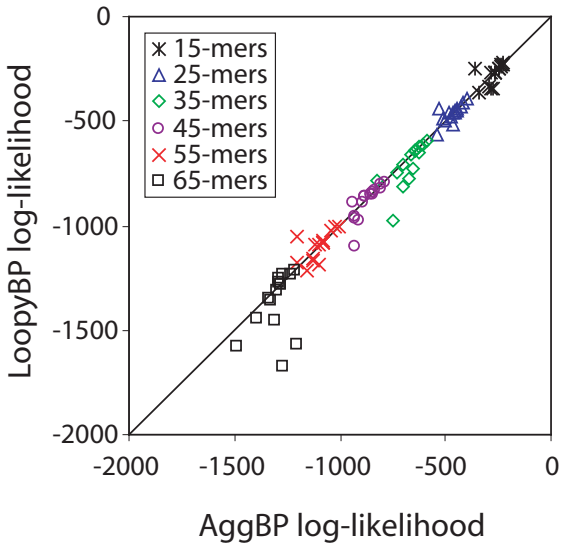


Figure 7.6 A scatterplot showing – for each of the 90 target fragments – the log likelihood of AGGBP's backbone model versus LOOPYBP's. Points below the diagonal correspond to fragments where AGGBP returns a solution with higher (where higher is better) likelihood.

7.4.2.1 Object generator

My object generator lets me vary the graph topology and individual part parameters shown in Figure 7.7. The generator constructs objects with a predefined number of parts, arranged in a skeleton with a predefined number of "skeletal loops". Given some branching factor, the skeleton is randomly assembled from the parts. As before, all pairs of parts not connected in the object skeleton are constrained to not occupy the same 3D space

Each part in the model is given a *radius* $r_i \in \mathbf{r}$ and a *softness* $s_i \in \mathbf{s}$, from which the structural potential functions are derived. Pairs of parts directly connected in the skeleton maintain a distance equal to the sum of their radii. Part orientation is not modeled (although it is not prohibited in the framework); only the distance between parts connected in the skeleton matters. All parts not connected in the skeleton should be *at least* as far apart as the sum of their radii (although the softness parameter allows this to be violated).

The softness parameter, which may be interesting in modeling objects with "compressible" parts, allows part pairs to get slightly closer than the sum of their radii with some low probability. Specifically, the softness parameter replaces the occupancy potential's step function with a sigmoid. For non-zero softness, then, the probability distribution of the distance d between two parts i and j, with radii r_i and r_j, and softness s_i and s_j, is given by:

$$p_{ij}(d) = \left(1 + exp\left(\frac{-(d - (r_i + r_j))}{s_i \cdot r_i + s_j \cdot r_j}\right)\right)^{-1} \tag{7.10}$$

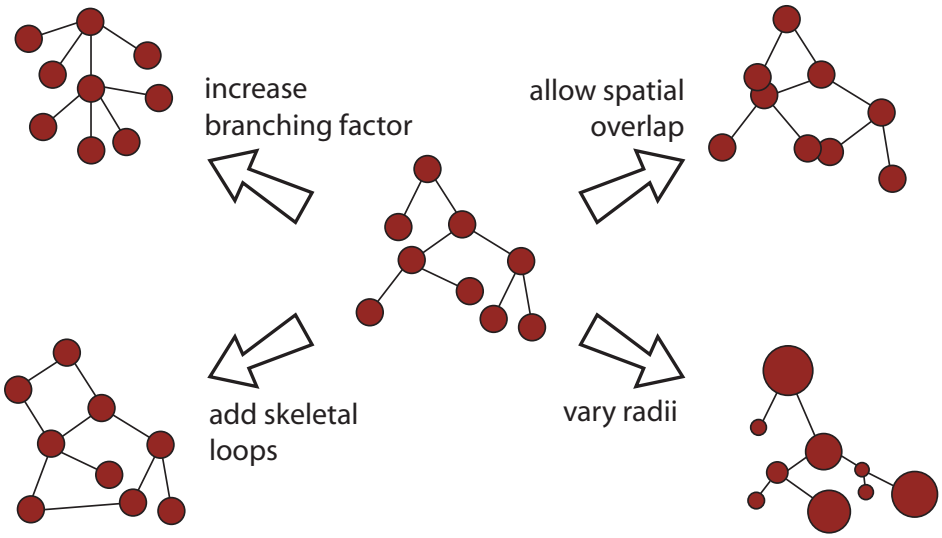


Figure 7.7 Four graph-topology parameters that one may vary using my graph generator.

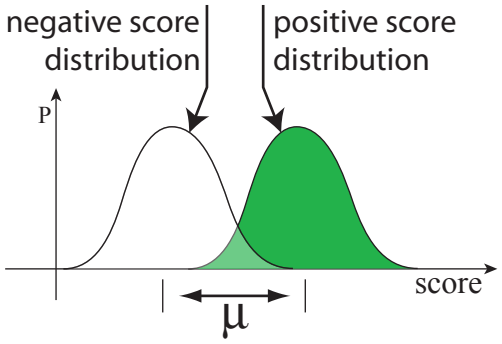


Figure 7.8 Observation potentials are generated by drawing scores from two distributions. The parameter μ is directly related to each part-classifier's accuracy.

The generator assumes that each part's radius and softness is known (or exactly learned from some training set) before inference.

My testbed generator also generates observation potentials ψ_{obs}, that is, the probability distribution of each part's location in 3D space. These would normally be generated by some type of pattern matcher in a 2D or 3D image. My generator assumes I have a classifier that – given a location in 3D space – returns a score. Shown in Figure 7.8, scores are drawn from one of two distributions: at the true location of a part, the score for that part is drawn from one distribution, at any other location the score is drawn from another distribution.

For simplification, I assume both distributions are fixed-width Gaussians with different means. Varying the difference in means results in classifiers with varying accuracy. Given this difference in means, then, I *generate* each part's observation potential by drawing "match-scores" at random from these two distribution. I assume the two distributions are known (or exactly learned from some training set); thus, I can convert match-score m_i into probabilities using Bayes' rule:

$$P(\textbf{pos-class}|m_i) = \frac{P(m_i|\textbf{pos-class}) \cdot P(\textbf{pos-class})}{P(m_i)}$$

$$= \frac{P(m_i|\textbf{pos-class}) \cdot P(\textbf{pos-class})}{P(m_i|\textbf{pos-class}) \cdot P(\textbf{pos-class}) + P(m_i|\textbf{neg-class}) \cdot P(\textbf{neg-class})} \quad (7.11)$$

The prior probability on the positive (and negative) class $P(\textbf{pos-class})$ is known, since I know there is one instance of each part in the image. The conditional probabilities $P(m_i|\textbf{pos-class})$ and $P(m_i|\textbf{neg-class})$ use the fact I know the distribution of positive-class and negative-class scores.

A specific width μ corresponds to a specific classifier accuracy. Greater values of μ mean the score distributions of the positive and negative classes are further apart, and thus more easily separated. One can think of building a classifier where, given some match-score m_i, one computes the probability as in Equation 7.11. Then, by varying the probability threshold which I use to separate positive and negative examples, one can generate a precision-recall curve. The area under this curve is one measure of classifier performance: higher values of μ correspond to greater area under this curve. In the remainder of this section, I report not this value for μ, but rather the area under the precision-recall curve (AUPRC) which it – along with the number of positive and negative examples – induces. For example, on a 40x40x40 grid, $\mu = 3.85$ corresponds to an AUPRC of 0.3.

Finally, there is no requirement that the skeleton of an object maintain a tree structure. The testbed generator allows construction of objects with a predefined number of loops in the skeleton graph.

7.4.2.2 Results

I use my generator to vary five different parameters (four topological) in the model (default values are shown in parentheses):

- **branching-factor**: the average branching factor in the skeleton graph (default = 2)
- **softness**: each part's softness (default = 0.0)
- $\boldsymbol{\sigma}$(radius): the standard deviation of radii in the graph (default = 0)
- $\boldsymbol{\mu}$: the difference in means between the positive score distribution and negative score distribution. I report this value as the area under the associated "part detector's" precision-recall curve. (default area = 0.3)
- **loop-count**: The number of loops in the skeleton graph. (default = 0)

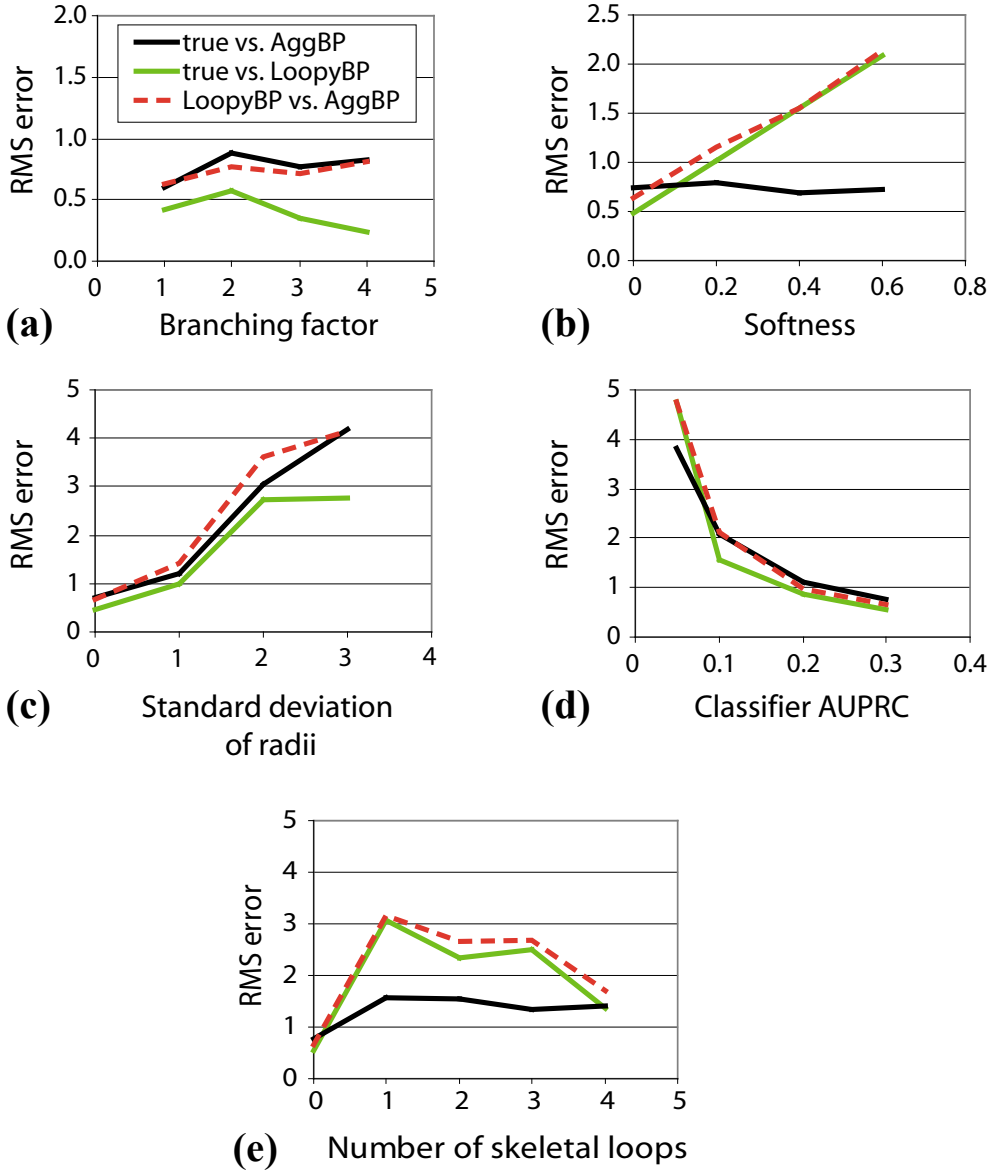


Figure 7.9 A comparison of AGGBP and LOOPYBP using the synthetic-object generator. While holding other parameters fixed, I vary (a) skeleton branching factor, (b) part softness, (c) radius standard deviation, (d) classifier AUPRC, and (e) skeletal-graph loop count. I report the RMS error of my algorithm (AGGBP) and LOOPYBP against each other as well as against ground truth.

In every graph, the average part radius is fixed, and each model is constructed of 100 parts.

I use my object-recognition framework to search for the optimal layout of parts, given some generated object and observation potentials. As in the Section 7.4.1, I use both standard loopy BP (LOOPYBP) and AGGBP, and compare the results. I assume that part parameters – radius and softness – are known (or learned) by the algorithm. For both AGGBP and LOOPYBP, I run until convergence, for a maximum of 20 iterations. In a few instances (typically with a poor "part detector"), BP did not converge; in these cases, I took the highest-likelihood solution at *any* iteration. At each parameter setting, I compute the average error using 20 randomly generated part graphs.

Results from this experiment appear in Figure 7.9. For each of the four varied parameters I plot the RMS error (a) between LOOPYBP and ground truth, (b) between AGGBP and truth, and (c) between AGGBP's solution and LOOPYBP's. The relative running time and memory usage of AGGBP compared to LOOPYBP follows the trend shown in Figure 7.3, with LOOPYBP taking approximately 5 times longer and using approximately 7 times the memory of AGGBP when locating a 100-part model.

For three of the varied parameters – graph branching factor, classifier AUPRC, and skeletal loops (Figure 7.9, Panels a, d and e) – the solutions returned by the two methods are of comparable accuracy. Even though the solutions themselves may be quite different, they are both equally close to ground truth. The odd behavior of the algorithm, as the number of skeletal loops increases (Panel e) – with error increasing for 1 and 2 loops, then decreasing for 3 and 4 – may be explained by the fact that the algorithm does not control the *length* of skeletal loops. With fewer loops in the skeletal graph, the average loop length is likely to be longer, which may cause problems for BP.

Figure 7.9c shows that my algorithm performs reasonably well as the radii of the model's parts are varied more. The performance of the two algorithms is similar until the standard deviation of part radii is increased to three times the radius. Larger variations could be handled by clustering objects into multiple groups based on radius, approximating messages to each group.

The most interesting result, however, is that in Figure 7.9b, where the object softness is varied. Increasing object softness allows two objects to move closer than would normally be allowed with some low probability. Here, for any non-zero softness, AGGBP finds a more accurate solution than LOOPYBP. The reason for this in unclear; it may be due to feedback introduced by this softness, that is dampened by AGGBP's approximation. When running with a non-zero softness, LOOPYBP often fails to converge, giving some support to the idea that my approximation is dampening some feedback loops.

Log likelihood plots, not shown for these synthetic experiments, are very similar to the error plots. These experiments show that AGGBP's approximation method is valid for a wide variety of model parameters and part topologies. In a large majority of the synthetic experiments, AGGBP produced an interpretation that was as good or better than LOOPYBP, in less time.

7.5 Conclusions and future work

I describe a part-based, 3D object recognition framework, well suited to analyzing detailed 3D image data. I introduce AGGBP, a message approximation and aggregation scheme that makes belief propagation tractable in large and highly connected graphs. Using a message-approximation similar to that of mean-field methods, I reduce the number of message computations at a single node from many to just a few. In the fully connected graphs used by my object-recognition framework, AGGBP reduces the running time and memory requirements for an object with N parts from O(N^2) to O(N).

Additionally, I describe an efficient probability representation based on Fourier series. Experiments on a 3D vision task arising from X-ray crystallography shows that using these improvements produce solutions as good or better then loopy BP. Synthetic tests show that AGGBP is accurate under a variety of object types with various part topologies, in most cases producing a solution as good or better than loopy BP.

It is unclear why AGGBP should sometimes produce more-accurate results than standard loopy BP. My approximate-message computation ignores a term that serves to avoid feedback, which makes LOOPYBP exact in tree-structured graphs. However, in graphs with loops, such feedback is unavoidable (through the loops of the graph). For some types of edge potentials, ignoring this term produces a more-accurate approximation, perhaps by dampening some of these feedback loops inherent in loopy belief propagation. Further investigation into this is needed.

In the future, I would like to investigate taking a more dynamic approach to message aggregation. For example, in the protein backbone-tracing task, AGGBP's approximation error is highest along edges connecting amino acids that are nearby in space (see Section B.2). If I could accurately predict which amino acids are close (in space) as BP iterates, I could *precisely* compute messages between these pairs of nodes, and *approximately* compute messages along other edges.

The results using AGGBP illustrate my techniques are useful in the automatic interpretation of complex 3D image data. The shortcuts I introduce drastically increase the size of problems on which BP is tractable. In one real and one synthetic dataset, I produce accurate results with significant CPU and storage savings over standard loopy BP. The algorithm presented in this chapter appears to be a powerful tool for locating complex objects in large images.

Chapter 8

Conclusion

My thesis describes several probabilistic techniques for automating the time-consuming interpretation of electron-density maps. With a growing need for high-throughput determination of protein structures, automation of this step is critically important. The methods outlined in this thesis have improved upon the state-of-the-art, allowing more accurate interpretation of poor-quality maps than other widely used approaches. An implementation of the algorithms described in my thesis is currently in regular use at the University of Wisconsin Center for Eukaryotic Structural Genomics.

Locating large, highly flexible proteins in large three-dimensional images requires the development of computational methods to handle statistical inference in a reasonable amount of time, while making a minimum number of simplifying assumptions. This work describes a novel message approximation scheme that allows inference to proceed in a reasonable amount of time, even with proteins containing more than 1000 amino acids. I derive a faster method for searching for template structures in the density map, using spherical-harmonic decomposition. Together, these advancements make the application of a complex statistical model tractable for this problem.

The probabilistic framework I use throughout this thesis is flexible and expressive, combining many local structural patterns into a global protein structure. As an illustration of its expressiveness, in Chapter 3, I describe how my algorithm can use the estimated location of selenium atoms (found from phasing experiments [46]) to adjust the prior probabilities on methionine, a selenium-containing amino acid. This thesis only scratches the surface of what is expressible in such a probabilistic framework, which cleanly allows the integration of many different sources of knowledge available to biologists. This includes sources such as distant structural analogues (or predicted structural analogues) [64], predicted secondary structure [98] and predicted contact maps [76]. Unlike any other approach, the probabilistic framework described in this thesis allows all these sources of information to be neatly integrated in a single model.

Finally, the model presented in this thesis is generalizable to other types of objects, from other image data sources, including fMRI scans [102] of the brain, high-quality 3D images of tissues produced by confocal microscopy [92], and detailed images of large macromolecular complexes produced by electron cryomicroscopy [12]. The general image framework presented in Chapter 7 makes application of my algorithm to any of these datasets relatively straightforward.

In each chapter of my thesis, I presented possible future research directions. Here, I present a summary of each chapter, including my contributions, as well as a summary of these future research directions.

8.1 Probabilistic protein-backbone tracing

Chapter 3 presented ACMI-BP, a tool for automatically tracing protein backbones especially designed for poor-quality electron-density maps. I designed a model-based approach to electron-density map interpretation. Modeling the protein using a pairwise Markov field, I predict the most likely layout *of a particular amino-acid sequence* given an electron-density map. The resultant backbone models are more accurate in terms of RMS error and model completeness then other automatic interpretation methods.

Future Work: One area where I believe significant performance gains are realizable is in improved methods for managing non-crystallographic symmetry [7]. Many maps have this type of symmetry, that is, where the protein forms a multimeric complex in the asymmetric unit. Every copy must be found by the automated method or the crystallographer, although often all copies are in identical or nearly identical conformations. Currently, ACMI-BP searches for all chains simultaneously, but does not use the knowledge that chains are likely to take similar conformations. This knowledge, however, is used by crystallographers in solving such maps, and is valuable in solving very poor density maps. In the future, I would like to add to ACMI the ability to infer the conformation of just a single protein chain and one or more transformations relating these multiple copies.

8.2 Improved template matching in density maps

Chapter 4 describes ACMI-SH, a significant improvement over previous work in 3D template matching in electron-density maps. Previous work by myself (Chapter 3's ACMI-FF) and others [17, 110] uses Fourier convolution to rapidly search a map. ACMI-SH considers the use of spherical-harmonic decomposition of a template to rapidly search all rotations of some fragment at a single (x, y, z) location. This method reduces computational time by allowing me to eliminate a majority of points from the map, only considering the 20% of points with the greatest density. This offers both improved efficiency and accuracy compared to previous work, finding substantially better models in approximately 60% of the running time.

Future Work: Template-matching requires a significant amount of CPU time, and represents the bottleneck in ACMI's interpretation pipeline (although parallelization alleviates this somewhat). An effort should be made in the future to address the high computational cost of ACMI-SH's template matching. One possible speedup may be achieved through further improvements in the first-pass filter. Even eliminating 80% of locations from the map, as I currently do, rotational alignment between fragments and map takes substantial computation time. Improved filtering would reduce the number of rotational alignments needed, significantly decreasing running time. One idea is to

use the rotation-invariant representation suggested by Kondor [67] to train a supervised classification algorithm, such as a support vector machine or artificial neural network to predict likely Cα locations in the density map. One could even build separate classifiers for each amino-acid type. This would allow for much more rapid fragment searching.

Another future direction made possible by this work involves integrating this template searching and probabilistic inference. Spherical-harmonic decomposition makes it possible to efficiently search for a fragment at a single location. This suggests an approach where, the initial probabilities are set using a coarse fragment search. Then, as inference proceeds, locations that appear to be good candidates for Cα's emerge, at which time a finer search of these locations could take place.

8.3 Constructing protein models using particle filtering

Chapter 5 builds upon the distributions computed in Chapter 3, using particle filtering to automatically produce a set of all-atom protein models given an electron-density map. Particle filtering considers growing an ensemble of all-atom protein models, at each iteration sampling the layout of an additional amino acid. A novel contribution of this work is that I use the inferred approximate marginal distributions (from Chapter 3) to grow the protein chains (or "particles"). In addition to improving the inferred backbone traces of Chapter 3, this approach allows one to generate multiple physically feasible structures that explain the density map, using multiple runs of particle filtering. I show that multiple structures – generated from multiple particle-filtering runs – do a better job than a single structure at explaining the density map.

Future Work: The idea of using multiple conformations to explain the observed map is an interesting one. Making use of these multiple conformations – even with just ten predicted structures, as I considered in Chapter 5 – is difficult, however. One promising research direction is to automatically present summarized information about the set of structures. For example, something akin to "In residues 34-39, 60% of structures take loop conformation A, while 40% take conformation B." In addition to better modeling of the observed data, this type of information may provide valuable information about atomic motion of the protein.

Another place where I believe significant gains are possible is in the use of more complicated reweighing functions. Chapter 3's BP inference limits the form that potential functions in the Markov-field model can take, as my empirical evidence suggests overly complicated potential functions tend to cause convergence problems. However, particle filtering does not suffer from this problem. Here, a reweighing function based on the physical energy (such as CHARMM's [9] force field) of the growing protein chain could better disambiguate unclear regions in the density map.

Finally, I presented some preliminary results the use of ACMI in an iterative fashion, where ACMI-PF models are used to improveme map quality; the improved maps are fed into the ACMI pipeline. Although I show that ACMI-PF is able to improve map quality, I fail to see these gains realized as improved 3D structural models. Further investigation into this is needed.

8.4 Atom-level matching using pictorial structures

Chapter 6 shows the application of Felzenszwalb and Huttenlocher's fast pictorial-structure matching algorithm [31] to locate individual sidechain atoms in an electron-density map. This chapter intoduces DEFT, which extends their algorithm in three key ways. In order to model atoms rotating in 3D, I designed a screw joint. I also developed extensions to deal with spatial collisions of parts in the pictorial structure model, and implemented an improved template-construction routine. The method works well at identifying individual atoms in medium-resolution maps; one strength – and corresponding weakness of ACMI – is that DEFT's flexible sidechain model is able to handle proteins with novel or rare sidechain conformations.

Future Work: One of the difficulties with this algorithm is in dealing with spatial collisions of parts. A different optimization algorithm, like loopy belief-propagation (as in Chapter 3), may be a more natural way of dealing with collisions. Additionally, incorporating this chapter's sidechain-matching approach into ACMI may have some accuracy benefit, and would be interesting to explore in the future.

8.5 A general object-recognition framework

Chapter 7 generalizes the approach I introduced in Chapter 3. I presented a part-based, 3D object-recognition framework. Inference in this model uses a novel message approximation and aggregation scheme that makes belief propagation tractable in large and highly connected graphs. In the fully connected graphs used by my object-recognition framework, the approximation reduces the running time and memory requirements for an object with N parts from O(N^2) to O(N).

Future Work: Although the message aggregation introduced in this chapter is intended purely for efficiency, in some cases aggregation leads to *better* solutions than standard loopy belief propagation. One interesting area of future research is in exploring why this is the case, and it what types of graphs this property holds.

Another future research direction involves taking a more dynamic approach to message aggregation. For example, in protein-backbone tracing, I see the greatest approximation error along edges connecting amino acids that are nearby in space (see Section B.2). If I can accurately predict which parts (that is, amino acids) are close (in space) as BP iterates, I could *precisely* compute messages between these pairs of nodes, and *approximately* compute messages along other edges.

8.6 Final wrapup

My thesis has shown that a probabilistic approach to density-map interpretation leads to the automatic construction of more-complete and more-accurate protein models than other automated approaches. A probabilistic approach has additional strengths: it has the ability to incorporate weak constraints that other methods are unable to employ. For example, Chapter 3 described a method to assist ACMI's backbone tracing by providing putative selenium locations. These selenium locations may be incorrect or missing, which makes it difficult for other methods to take advantage

of this information, but for ACMI using this "fuzzy information" is straightforward. In addition, in my probabilistic framework, it is straightforward to produce ensembles of protein models that better explain the observed density. There is some evidence [11, 74] that multiple conformations may better model natural structural variation within the protein crystal. Finally, my approach is modular; it is simple to plug in an improved matching algorithm, or an improved particle-filtering reweighing function.

The work presented here further extends the resolution of density maps that can be automatically interpreted, and allows accurate protein-structure determination on lower quality data. This helps speed the process of structure acquisition, allows lower-cost acquisition of protein structures, and enables interpretation of proteins for which good crystals may not be grown.

Appendix A: Datasets

This appendix summarizes the datasets of electron density maps used for experiments throughout this thesis. PDB accession codes [3] are included, as well as properties of each density map and each protein contained within the maps.

A.1 Dataset 1

Dataset 1 consists of four of the first five X-ray crystallography structures solved by the Center for Eukaryotic Structural Genomics (CESG) at the University of Wisconsin. This dataset was used for training and testing in Chapter 6's experiments. An overview of the dataset is shown in Table A.1.

Experiments using this data employed *intermediate* phase information; that is, phases were taken from a point where the crystallographer had built some, but not all, of the molecular model.

These four density maps were each *downsampled* to 2.5, 3, and 4Å resolution by dropping all higher-resolution reflections (that is, reflection intensity was zeroed), and recomputing the density map.

A.2 Dataset 2

My second dataset, used as the training and testing set for Chapters 3, 4, and 7, consists of ten X-ray crystallography structures solved by CESG. This dataset is a snapshot of CESG-solved crystal structures "captured" at a later timepoint than Dataset 1. To allow experiments to finish in a reasonable amount of time, I randomly selected ten maps from the 24 maps solved by CESG at this point in time. An overview of the dataset is shown in Table A.2.

For these ten maps, *model* phasing was used, that is, the phases from the final refined model were used to construct the electron-density map.

Each of these ten maps were downsampled to $R_0 = 2.5$, 3, 3.5, and 4Å resolution. To provide a more-realistic model of poor-resolution maps, as well as to avoid truncation effects, the maps were downsampled by *smoothly* diminishing the reflection intensities: I scaled the measured structure factors by $exp(-K/R^2)$, where R is the resolution of the structure factor and K is a scaling

Table A.1 The four proteins in Dataset 1.

PDB id	Residues in asymm. unit	Molecules in asymm. unit	Molecules in unit cell	Resolution (Å)	Unit cell size (Å)
1Q44	326	1	8	1.9	91×121×74
1Q45	782	2	8	2.0	78×85×122
1Q4M	442	2	16	2.1	63×63×288
1Q4R	112	1	6	1.9	55×55×58

Table A.2 The ten proteins in Dataset 2.

PDB id	Residues in asymm. unit	Molecules in asymm. unit	Molecules in unit cell	Resolution (Å)	Unit cell size (Å)
1Q4R[a]	112	1	6	1.9	55×55×58
1VJH	244	2	4	2.1	46×34×79
1VK5	157	1	6	1.7	83×83×61
1VMO	260	2	8	1.9	60×79×44
1XFI	367	1	1	1.7	40×43×53
1XM8	508	2	4	1.8	68×59×69
1XMT	103	1	2	2.0	27×61×29
1XQ1	266	1	8	2.1	56×77×112
1XY7	332	2	12	1.8	56×56×147
1YDH	432	2	8	2.3	122×80×51

[a] Also appears in Testset 1.

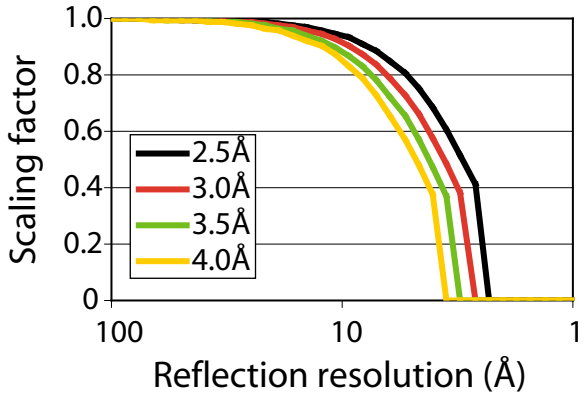


Figure A.1 Data was downsampled by smoothly diminishing reflection intensities. A reflection with resolution indicated on the x-axis had its intensity scaled by the corresponding value on the y-axis.

constant chosen based on the desired resolution (higher values of K smooth the map more). I chose $K = R_0^2$, so the signal strength was weakened by $1/e$ at the point of truncation. Figure A.1 shows how I scale structure factors as a function of resolution.

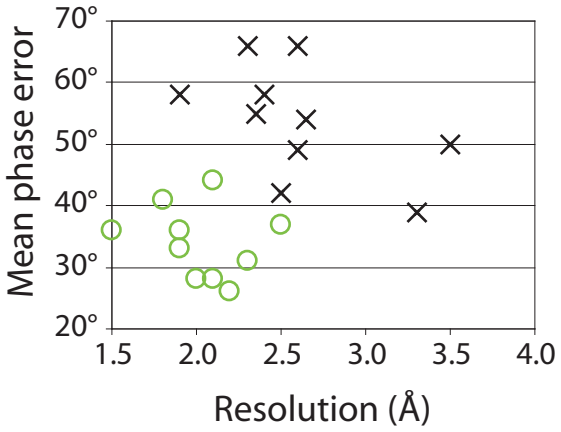


Figure A.2 A scatterplot showing (raw) experimental data quality. For each of the twenty maps considered for inclusion, the x-axis indicates (phased) map resolution, while the y-axis shows the mean phase error. The $\times$'s indicate the ten "poor-quality" maps chosen for inclusion.

A.3 Dataset 3

My final dataset – used as a testset in Chapter 5 – was also provided by the Center for Eukaryotic Structural Genomics (CESG) at UW–Madison. This dataset used experimental intensities and with the initial phasing (typically obtained using SAD [46] or MAD [44]) available to the crystallographer at the start of model-building. The maps were initially phased using AUTOSHARP [110], with non-crystallographic symmetry averaging used to improve the map quality where possible. The ten maps were selected as the "most difficult" from a larger dataset of twenty maps provided by CESG. The difficulty assessment of these maps was based on expert judgment of the electron-density quality (by collaborator E. Bitto), as well as quantitative estimate of phase error. These structures have been previously solved and deposited to the PDB, enabling a direct comparison with the final refined model. All ten required a great deal of human effort to build and refine the final atomic model.

The data are summarized in Table A.3, with quality described by the resolution and phase error. The resolution refers to that available from the initial phasing, which may not have reached the resolution limit of the data set. The initial low-resolution phasing was computationally extended in three structures to higher-resolution shells, using the algorithm implemented in RESOLVE. The mean phase error was computed by comparing the phases calculated from the final all-atom model with those in the initially phased data set. The CCP4 [13] suite of programs was used to perform these calculations.

Finally, for this dataset, since phasing quality plays an important role, this table also reports the mean phase error of the experimental phases versus the phases in the final, refined model. Figure A.2 illustrates our dataset's coverage of resolution-phase error space. All twenty maps

Table A.3 The ten proteins in Dataset 3.

PDB id	Residues in ASU	Molecules in ASU	Molecules in unit cell	Resolution (Å)	Mean phase error [b]	Unit cell size (Å)
2NXF[b]	322	1	8	1.9	58°	64×87×157
2Q7A[b]	316	2	12	2.6	49°	82×82×107
XXXX[d]	566	2	16	2.65	54°	119×119×84
1XRI	430	2	24	3.3	39°	124×124×124
1ZTP	753	3	12	2.5	42°	63×117×124
1Y0Z	660	2	8	2.4 (3.7[c])	58°	145×61×115
2A3Q	340	2	24	2.3 (3.5[c])	66°	74×74×236
2IFU	1220	4	16	3.5	50°	84×91×265
2BDU	594	2	6	2.35	55°	134×134×39
2AB1	244	2	8	2.6 (4.0[c])	66°	46×58×89

[a] Averaged over all resolution shells.
[b] Different dataset was used to solve the PDB structure.
[c] Phasing was extended from worse resolution.
[d] PDB file not yet released.

considered for inclusion are shown in this scatterplot; the ten chosen for inclusion are illustrated with ×'s. The chosen maps had worse-than-average resolution or phase error (or both), and were verified by crystallographers who worked on these maps as difficult maps to interpret.

Appendix B: Supplementary experiments

I present in this appendix some supplementary experimental data, outside of the scope of their respective chapters. The first experiment provides some support for my decision to model probability distributions in rotational space using a single Gaussian. In the second experiment, I expand on the experiments of Chapter 7, showing AGGBP's approximation errors are greatest when between Cα's that are close (in Cartesian space) in the final deposited protein structure.

B.1 Modeling probabilities in rotational-space

In Section 3.3.1.1, I describe how ACMI models probability distributions in rotational space. Specifically, that section describes how I – in an effort to reduce storage computational complexity – model probabilities in rotational space by storing at each location (x, y, z), a single orientation $(\theta_b, \phi_b, \theta_f, \phi_f)$, and assume probabilities are distributed as a Gaussian around this stored value. I hypothesized that storing probability distributions in this manner would lead to a minor loss in accuracy.

Before making this approximation, I ran a set of experiments to gauge the validity of this hypothesis. I randomly sample regions of density around 500 Cα atoms in 10 different electron-density maps (Chapter 3's dataset; see Section A.2). Around each Cα location (x_i, y_i, z_i), I compute the *full* observation potential ψ_i as a function of the rotational parameters. I consider modeling these distributions in rotational space using a mixture of Gaussians. The mixture components are fit greedily: the first Gaussian is fit to the distribution, the second is fit to the residual of the single-Gaussian model, the third to the two-Gaussian residual, and so on. To avoid overfitting, no two mixture components may be closer than 15°. The standard deviation of each mixture component was fixed at 1Å.

The results are shown in Figures B.1 and B.2. Recall that observation potentials ψ_i are constructed as the weighted sum of match probabilities from each of a set of templates. Figure B.1 compares the observation potential to the mixture-of-Gaussian approximation for a single template. In the plot, the x axis shows the number of mixture components; that is, the number of stored orientations at each position (x_i, y_i, z_i). The y-axis shows the correlation coefficient between a single template's contribution to ψ_i versus the N-Gaussian approximation. Figure B.2 plots the same values; here ψ_i is computed as it is in ACMI, as a weighted sum of individual template probabilities.

As these plots show, most of the correlation is gained with the first mixture component; additional components provide minimal error reduction. This is especially evident in the multi-component ψ_i. A single template's contribution to ψ_i is modeled very well using a single Gaussian, with average correlation coefficient over 0.4. However, for the complete ψ_i, which is comprised of the weighted contribution of around 50 templates, I see correlation coefficients closer to 0.2.

I suspect the reason for the reduced correlation is that there is significant high frequency noise in the complete ψ_i. This seems verified by the fact that additional Gaussian components after the first, at least 15° away, improve correlation coefficients very little. This noise is not particularly important to model. Additional support for the use of a single Gaussian come from the observation

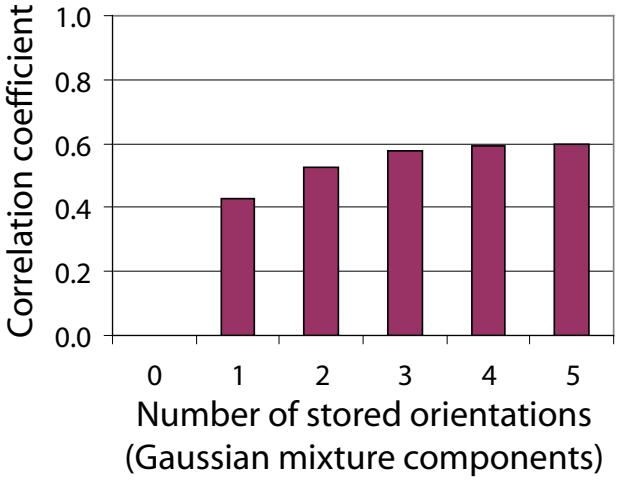


Figure B.1 A comparison of a single template's contribution to the observation potential ψ_i to the mixture-of-Gaussian approximation. the x axis shows the number of mixture components; the y-axis shows the correlation coefficient between a single template's contribution to ψ_i versus the N-Gaussian approximation.

that, when modeling the complete ψ_i, the peak of the single stored Gaussian is on average only 18° from the true peak. For these reasons, I feel a single Gaussian is sufficient to accurately model probability distributions in rotational space.

B.2 Message-approximation errors as a function of Cα–Cα distance

In Section 7.5, I note that AGGBP's occupancy message errors are greatest in messages between amino acids that are close in Cartesian space, in the "ground truth" model. This section provides experimental support for this statement. Using only the 65-amino-acid long fragments from Section 7.4.1's dataset, I compute Kullback-Leibler (KL) divergence [70] between AGGBP and LOOPYBP's messages as a function of the Cα–Cα distance between amino-acids sending and receiving the message. To avoid the additive effect of message errors, I only consider messages passed during the first iteration of AGGBP/LOOPYBP.

Figure B.3 shows the results of this experiment. This figure shows that the average message's KL divergence is approximately 50% greater between amino acids that are close (¡10Å between Cα's) than it is between those that are distant (¿14Å). Thus, a dynamic message aggregation scheme, as mentioned in 7.5 may be beneficial at reducing approximation errors.

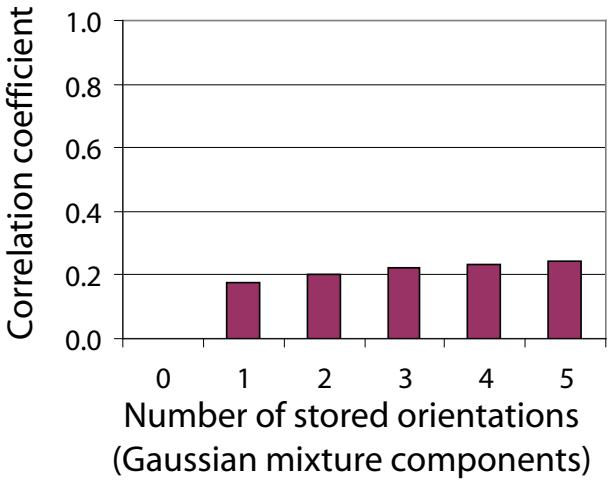


Figure B.2 A comparison of ACMI's observation potential ψ_i to the mixture-of-Gaussian approximation. Axes are the same as in Figure B.1

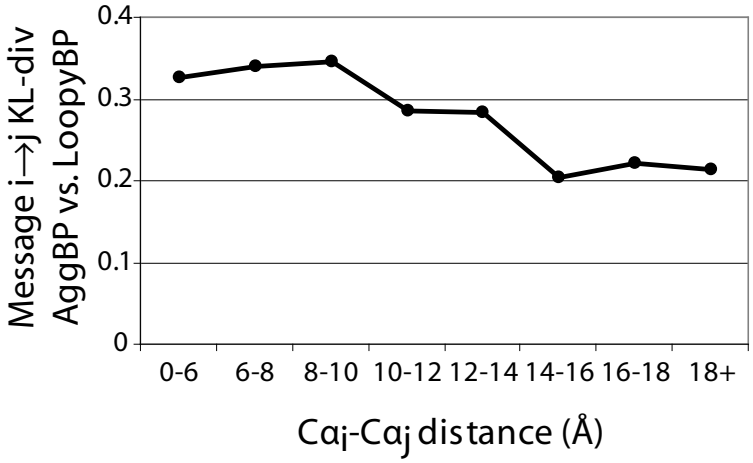


Figure B.3 The KL-divergence of AGGBP's messages versus LOOPYBP's messages as a function of Cα–Cα distance. The x-axis bins the Cα–Cα distance between amino acids, while the y-axis shows the average KL-divergence between AGGBP and LOOPYBP messages between amino acids in the corresponding distance bin.

GLOSSARY

alpha carbon (or Cα)
A central atom in each amino acid that forms the interface between *backbone* and *sidechain*. A *backbone trace* identifies the location of only Cα atoms.

amino-acid residue (or *residue*)
The portion of an amino acid that remains after condensation into a polypeptide chain. Proteins are formed when amino acids condense to form a chain of amino-acid residues, connected by peptide bonds.

backbone
The repeating four (non-hydrogen) atom motif in a polypeptide chain.

conditional probability
The probability of some event A, *given* that some other event B has occurred. The conditional probability is denoted $P(A|B)$, which is read "the probability of A, given B."

electron-density map (or *density map*, *map*)
A three-dimensional "picture" of the electron clouds surrounding each protein atom. The density map is generated in protein crystallography, as the Fourier transform of the complex-valued reflections.

likelihood
In statistics, a likelihood function is function of the second argument of a conditional probability, with the first argument fixed. In this thesis, likelihood is often used synonymously with probability.

marginal probability
Given random variables A, B, and C, the marginal probability of some variable A is the probability of A ignoring the settings of B and C. It is calculated by summing the joint probability distribution over all variables except A. That is, given the joint distribution $P(A, B, C)$, one computes the marginal distribution of A as $P_A(A) = \sum_B \sum_C P(A, B, C)$, with sums over all possible outcomes of events B and C.

Markov network (or *Markov random field*, *MRF*)
A model of the full-joint probability of a set of random variables, where the probability of some setting of the variables is defined on an undirected graph, as the product of *potential functions* associated with each clique (i.e., fully connected subgraph) of the graph.

pairwise Markov network (or *pairwise Markov random field*)
A *Markov network* with no potential functions of > 2 variables; the only potential functions are those associated with edges and vertices in the undirected graph.

phase (see also *reflection*, *resolution*)
When (complex-valued) reflection data is collected in X-ray crystallography, only *intensities* are measurable. Phases must be approximated using some other method. This problem is known as the *phase problem* and is solved through several different experimental and computational techniques.

R **factor**
A crystallographic model-evaluation metric. The R factor is statistical residual measure of the deviation between the reflection intensities predicted by the model and those experimentally measured. Crystallographic R factors typically range from 0.0 (indicating a perfect fit of the model to the density map) to 0.6 (the score of a random model). R factors are typically 0.2 or less in solved structures.

R_{free} (see also R *factor*, R_{work})
The R factor on a testset of held-aside reflections, to avoid model overfitting.

R_{work} (see also R *factor*, R_{free})
The R factor on the "training set," that is, on all reflections not used in calculation of R_{free}.

reflection
In crystallography, the name given to each of the spots formed when a protein crystal diffracts an X-ray beam. An electron-density map is computed as the Fourier transform of these complex-valued reflections. However, only reflection intensities are measurable; *phases* must be approximated using some other method.

resolution (see also *reflection*, *phase*)
Each measured reflection in a crystal's diffraction pattern has a corresponding resolution, which measures the interplanar spacing of that reflection's contribution to the density map. The resolution of a density map is the minimal such spacing over all reflections.

sidechain
The variable region in each of the twenty naturally occurring amino acids.

trace (or *backbone trace*, *C*α *trace*)
A model of a protein where the location of only one atom in each amino-acid residue, the alpha carbon, is identified.

LIST OF REFERENCES

[1] J. Abrahams and R. De Graaff. New developments in phase refinement. *Current Opinion in Structural Biology*, 8(5):601–605, 1998.

[2] M.S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear/non-Gaussian Bayesian tracking. *IEEE Transactions of Signal Processing*, 50:174–188, 2001.

[3] H. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. Bhat, H. Weissig, I. Shindyalov, and P. Bourne. The protein data bank. *Nucleic Acids Research*, 28:235–242, 2000.

[4] J. Besag and J. York. *Analysis of Statistical Information*, pages 491–507. Bayesian restoration of images. Institute of Statistical Mathematics, 1989. *Ed.*: T. Matsunawa.

[5] C. Bishop. Variational principle components. In *Proceedings of the Ninth International Conference on Artificial Neural Networks*, pages 509–514, Edinburgh, Scotland, 1999.

[6] C. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[7] D. Blow. *Noncrystallographic Symmetry*, volume F of *International Tables for Crystallography*, pages 263–268. Springer, 2006.

[8] D. Blow and M. Rossmann. The single isomorphous replacement method. *Acta Crystallographica*, 14:1195–1202, 1961.

[9] B. Brooks, R. Bruccoleri, B. Olafson, D. States, S. Swaminathan, and M. Karplus. CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4:187–217, 1983.

[10] A. Brunger. Free R value: A novel statistical quantity for assessing the accuracy of crystal structures. *Nature*, 355:472–475, 1992.

[11] F. Burling and A. Brunger. Thermal motion and conformational disorder in protein crystal-structures – comparison of multi-conformer and time-averaging models. *Israel Journal of Chemistry*, 34:165–175, 1994.

[12] W. Chiu. What does electron cryomicroscopy provide that X-ray crystallography and spectroscopy cannot? *Annual Review of Biophysics and Biomolecular Structure*, 22:233–255, 1993.

[13] Collaborative Computational Project, Number 4. The CCP4 suite: Programs for protein crystallography. *Acta Crystallographica*, D50:760–763, 1994.

[14] J. Coughlan and S. Ferreira. Finding deformable shapes using loopy belief propagation. In *Proceedings of the Seventh European Conference on Computer Vision*, pages 453–468, Copenhagen, Denmark, 2002.

[15] K. Cowtan. Picture book of fourier. `http://www.ysbl.york.ac.uk/~cowtan/fourier/fourier.html`.

[16] K. Cowtan. Modified phased translation functions and their application to molecular-fragment location. *Acta Crystallographica*, D54:750–756, 1998.

[17] K. Cowtan. Fast Fourier feature recognition. *Acta Crystallographica*, D57:1435–1444, 2001.

[18] K. Cowtan. The Clipper C++ libraries for X-ray crystallography. *IUCr Computing Commission Newsletter*, 2:4–9, 2003.

[19] D. Cromer and J. Mann. X-ray scattering factors computed from numerical hartree-fock wave functions. *Acta Crystallographica*, A24:321–324, 1968.

[20] R. Crowther. *The Molecular Replacement Method*. Gordon and Breach, 1972.

[21] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39:1–38, 1977.

[22] M. DePristo, P. de Bakker, and T. Blundell. Heterogeneity and inaccuracy in protein structures solved by X-ray crystallography. *Structure*, 12:911–917, 2004.

[23] F. DiMaio, D. Kondrashov, E. Bitto, A. Soni, C. Bingman, G. Phillips Jr., and J. Shavlik. Creating protein models from electron-density maps using particle-filtering methods. *Bioinformatics*, 2007. *In press*.

[24] F. DiMaio and J. Shavlik. Belief propagation in large, highly connected graphs for 3D part-based object recognition. In *Proceedings of the Sixth IEEE International Conference on Data Mining*, pages 845–850, Hong Kong, 2006.

[25] F. DiMaio and J. Shavlik. Improving the efficiency of belief propagation in large, highly-connected graphs, 2006. *UW ML Research Group Working Paper* 06-1.

[26] F. DiMaio, J. Shavlik, and G. Phillips. Pictorial structures for molecular modeling: Interpreting density maps. In *Advances in Neural Information Processing Systems*, pages 369–376, 2004.

[27] F. DiMaio, J.W. Shavlik, and G.N. Phillips Jr. A probabilistic approach to protein backbone tracing in electron-density maps. *Bioinformatics*, 22:e81–e89, 2006.

[28] F. DiMaio, A. Soni, G.N. Phillips Jr., and J.W. Shavlik. Improved methods for template-matching in electron-density maps using spherical harmonics. In *Proceedings of the IEEE Conference on Bioinformatics and Biomedicine*, Fremont, California, 2007.

[29] A. Doucet, S. Godsill, and C. Andrieu. On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10:197–208, 2000.

[30] L. Evans. *Partial Differential Equations*. The Laplace equation. American Mathematical Society, 1998.

[31] P. Felzenszwalb and D. Huttenlocher. Efficient matching of pictorial structures. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 66–73, Hilton Head, South Carolina, 2000.

[32] P. Felzenszwalb and D. Huttenlocher. Efficient belief propagation for early vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 261–268, Washington, DC, 2004.

[33] M. Fischler and R. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, 22:67–92, 1973.

[34] B. Frey. *Graphical Models for Machine Learning and Digital Communication*. MIT Press, 1998.

[35] N. Furnham, T. Blundell, M. DePristo, and T. Terwilliger. Is one solution good enough? *Nature Structural and Molecular Biology*, 13:184–185, 2006.

[36] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:721–741, 1984.

[37] W. Gilks and C. Berzuini. Following a moving target – Monte Carlo inference for dynamic Bayesian models. *Journal of the Royal Statistical Society*, 63:127–146, 2001.

[38] K. Gopal, T. Romo, E. Mckee, K. Childs, L. Kanbi, R. Pai, J. Smith, J. Sacchettini, and T. Ioerger. TEXTAL: Automated crystallographic protein structure determination. In *Proceedings of the Seventeenth Innovative Applications of Artificial Intelligence Conference*, pages 1483–1490, Pittsburgh, Pennsylvania, 2005.

[39] K. Gopal, T. Romo, J. Sacchettini, and T. Ioerger. Weighting features to recognize 3D patterns of electron density in X-ray protein crystallography. In *Proceedings on the IEEE Conference on Computational Systems Bioinformatics*, pages 255–265, Stanford, California, 2004.

[40] J. Greer. Three-dimensional pattern recognition: An approach to automated interpretation of electron density maps of proteins. *Journal of Molecular Biology*, 82:279–301, 1974.

[41] H. Groemer. *Geometric Applications of Fourier Series and Spherical Harmonics*. Cambridge University Press, 1996.

[42] W. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57:97–109, 1970.

[43] D. Healy, D. Rockmore, P. Kostelec, and S. Moore. FFTs for the 2-sphere – improvements and variations. *Journal of Fourier Analysis and Applications*, 9:341–85, 2003.

[44] W. Hendrickson. Maturation of MAD phasing for the determination of macromolecular structures. *Journal of Synchrotron Radiation*, 6:845–851, 1999.

[45] W. Hendrickson and E. Lattman. Representation of phase probability distributions for simplified combination of independent phase information. *Acta Crystallographica*, B26:136–143, 1970.

[46] W. Hendrickson and M. Teeter. Structure of the hydrophobic protein crambin determined directly from the anomalous scattering of sulphur. *Nature*, 290:107–113, 1981.

[47] T. Heskes. On the uniqueness of loopy belief propagation fixed points. *Neural Computation*, 16:2379–2413, 2004.

[48] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *International Journal of Approximate Reasoning*, 15(3):225–263, 1996.

[49] H. Huang, L. Shen, R. Zhang, F. Makedon, B. Hettleman, and J. Pearlman. Surface alignment of 3D spherical-harmonic models: Application to cardiac MRI analysis. In *Proceedings of the Eighth International Conference on Medical Image Computing and Computer Assisted Intervention*, pages 67–74, Palm Springs, California, 2005.

[50] S. Huang and J. Hwang. Computation of conformational entropy from protein sequences. *Proteins*, 59(4):802–809, 2004.

[51] D. Husmeier, R. Dybowski, and S. Roberts, editors. *Probabilistic Modelling in Bioinformatics and Medical Informatics*. Springer, 2004.

[52] A. Ihler, E. Sudderth, W. Freeman, and A. Willsky. Efficient multiscale sampling from products of Gaussian mixtures. In *Advances in Neural Information Processing Systems*, pages 1–8, 2004.

[53] T. Ioerger, T. Holton, J. Christopher, and J. Sacchettini. TEXTAL: A pattern recognition system for interpreting electron density maps. In *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*, pages 130–137, Heidelberg, Germany, 1999.

[54] T. Ioerger and J. Sacchettini. Automatic modeling of protein backbones in electron density maps via prediction of C-alpha coordinates. *Acta Crystallographica*, D58:2043–2054, 2002.

[55] T. Ioerger and J. Sacchettini. The TEXTAL system: Artificial intelligence techniques for automated protein model building. *Methods in Enzymology*, 374:244–270, 2003.

[56] M. Isard. PAMPAS: Real-valued graphical models for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 613–620, Madison, Wisconsin, 2003.

[57] S. Johnson. Hierarchical clustering schemes. *Psychometrika*, 2:241–254, 1967.

[58] D. Jones, W. Taylor, and J. Thornton. The rapid generation of mutation data matrices from protein sequences. *Computer Applications in the Biosciences*, 8:275–282, 1992.

[59] T. Jones, J. Zou, S. Cowan, and M. Kjeldgaard. Improved methods for building protein models in electron density maps and the location of errors in these models. *Acta Crystallographica*, A47:110–119, 1991.

[60] M. Jordan, editor. *Learning in Graphical Models*. MIT Press, 1998.

[61] M. Jordan, Z. Ghahramani, T. Jaakkola, and L. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233, 1999.

[62] A. MacKerell Jr., J. Wiorkiewicz-Kuczera, and M. Karplus. An all-atom empirical energy function for the simulation of nucleic acids. *Journal of the American Chemical Society*, 117:11946–11975, 1995.

[63] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica*, A32:922–923, 1976.

[64] D. Kim, D. Chivian, and D. Baker. Protein structure prediction and analysis using the Robetta server. *Nucleic Acids Research*, 32 (Supplement 2):W526–W531, 2004.

[65] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[66] D. Koller, U. Lerner, and D. Angelov. A general algorithm for approximate inference and its application to hybrid Bayes nets. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 324–333, Stockholm, Sweden, 1999.

[67] R. Kondor. A complete set of rotationally and translationally invariant features for images. *Preprint*, arXiv:cs/0701127v2, 2007.

[68] A. Kong, J. Liu, and W. Wong. Sequential imputations and Bayesian missing data problems. *Journal of the American Statistical Association*, 89:278–288, 1994.

[69] P. Kostelec and D. Rockmore. FFTs on the rotation group, 2003. *Working Paper Series, Santa Fe Institute* 03-11-060.

[70] S. Kullback and R. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:79–86, 1951.

[71] V. Lamzin and K. Wilson. Automated refinement of protein models. *Acta Crystallographica*, D49:129–147, 1993.

[72] M. Lee and I. Cohen. Proposal maps driven MCMC for estimating human body pose in static images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 334–341, Washington, DC, 2004.

[73] L. Leherte, J. Glasgow, K. Baxter, E. Steeg, and S. Fortier. Analysis of three-dimensional protein images. *Journal of Artificial Intelligence Research*, 7:125–159, 1997.

[74] E. Levin, D. Kondrashov, G. Wesenberg, and G. Phillips Jr. Ensemble refinement of protein crystal structures: Validation and application. *Structure*, 2007. *In press*.

[75] D. Levitt. A new software routine that automates the fitting of protein X-ray crystallographic electron density maps. *Acta Crystallographica*, D57:1013–1019, 2001.

[76] R. MacCallum. Striped sheets and protein contact prediction. *Bioinformatics*, 20 (Supplement 1):I224–I231, 2004.

[77] D. MacKay and R. Neal. Good codes based on very sparse matrices. In *Proceedings of the Fifth Conference on Cryptography and Coding*, pages 100–111, Cirencester, UK, 1995.

[78] R. McEliece, D. MacKay, and J. Cheng. Turbo decoding as an instance of Pearl's "belief propagation" algorithm. *IEEE Journal on Selected Areas in Communications*, 16(2):140–152, 1998.

[79] D. McRee. XtalView/Xfit–a versatile program for manipulating atomic coordinates and electron density. *Journal of Structural Biology*, 125(2-3):156–165, 1999.

[80] J. Mitchell. Uniform distributions of 3D rotations. *Unpublished document*, 2002.

[81] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.

[82] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[83] R. Morris, A. Perrakis, and V. Lamzin. ARP/wARP's model-building algorithms: The main chain. *Acta Crystallographica*, D58:968–975, 2002.

[84] R. Morris, A. Perrakis, and V. Lamzin. ARP/wARP and automatic interpretation of protein electron density maps. *Methods in Enzymology*, 374:229–244, 2003.

[85] K. Murphy, Y. Weiss, and M. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, pages 467–475, Stockholm, Sweden, 1999.

[86] G.N. Murshudov, A.A. Vagin, and E.J. Dodson. Refinement of macromolecular structures by the maximum-likelihood method. *Acta Crystallographica*, D53:240–255, 1997.

[87] C. Musso, N. Oudjane, and F. LeGlan. *Sequential Monte Carlo Methods in Practice*, pages 247–271. Improving regularised particle filters. Springer-Verlag, 2001. *Eds.*: A. Doucet, J. de Freitas, and N. Gordon.

[88] R. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[89] R. Neal and G. Hinton. *Learning in Graphical Models*, pages 355–368. A view of the EM algorithm that justifies incremental, sparse, and other variants. Kluwer, 1998. *Ed.*: M. Jordan.

[90] T. Oldfield. A number of real-space torsion-angle refinement techniques for proteins, nucleic acids, ligands and solvent. *Acta Crystallographica*, D57:82–94, 2001.

[91] L. Palmer, K. Scurrah, M. Tobin, S. Patel, J. Celedon, P. Burton, and S. Weiss. Genome-wide linkage analysis of longitudinal phenotypes using sigma2a random effects (SSARs) fitted by Gibbs sampling. *BMC Genetics*, 4(Supplement 1):S12, 2003.

[92] J. Pawley, editor. *Handbook of Biological Confocal Microscopy*. Springer, 3rd edition, 2006.

[93] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufman, 1988.

[94] A. Perrakis, T. Sixma, K. Wilson, and V. Lamzin. wARP: Improvement and extension of crystallographic phases by weighted averaging of multiple refined dummy atomic models. *Acta Crystallographica*, D53:448–455, 1997.

[95] C. Peterson and J. Anderson. A mean field theory learning algorithm for neural networks. *Complex Systems*, 1:995–1019, 1987.

[96] M. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filter. *Journal of the American Statistical Association*, 94:590–599, 1999.

[97] G. Rhodes. *Crystallography Made Crystal Clear*. Academic Press, 2000.

[98] B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, 232:584–599, 1993.

[99] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, 2002.

[100] T. Sawasaki, T. Ogasawara, R. Morishita, and Y. Endo. A cell-free protein synthesis system for high-throughput proteomics. *Proceedings of the Nataional Academy of Sciences*, 99:14652–14657, 2002.

[101] B. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall, 1986.

[102] S. Smith. Overview of fMRI analysis. *British Journal of Radiology*, 77:S167–S175, 2004.

[103] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[104] G. Snell, C. Cork, R. Nordmeyer, E. Cornell, G. Meigs, D. Yegian, J. Jaklevic, J. Jin, R. Stevens, and T. Earnest. Automated sample mounting and alignment system for biological crystallography at a synchrotron source. *Structure*, 12:537–545, 2004.

[105] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 605–612, Madison, Wisconsin, 2003.

[106] E. Sudderth, M. Mandel, W. Freeman, and A. Willsky. Visual hand tracking using nonparametric belief propagation. Technical Report 2603, MIT LIDS, 2004.

[107] E. Sudderth, M. Mandel, W. Freeman, and A Willsky. Distributed occlusion reasoning for tracking with nonparametric belief propagation. In *Advances in Neural Information Processing Systems*, pages 1369–1376, 2005.

[108] C. Sutton and A. McCallum. Improved dynamic schedules for belief propagation. In *Proceedings of the 23rd Conference on Uncertainty in Artificial Intelligence*, Vancouver, BC, Canada, 2007.

[109] S. Tatikonda and M. Jordan. Loopy belief propagation and gibbs measures. In *Proceedings on the Eighteenth Conference in Uncertainty in Artificial Intelligence*, pages 493–500, Edmonton, Alberta, Canada, 2002.

[110] T. Terwilliger. Automated mainchain model-building by template-matching and iterative fragment extension. *Acta Crystallographica*, D59:38–44, 2002.

[111] T. Terwilliger. Automated sidechain model-building and sequence assignment by template-matching. *Acta Crystallographica*, D59:45–49, 2002.

[112] D. Thain, T. Tannenbaum, and M. Livny. Distributed computing in practice: The Condor experience. *Concurrency and Computation*, 17:323–356, 2005.

[113] S. Trapani and J. Navaza. Calculation of spherical harmonics and Wigner d functions by FFT. *Acta Crystallographica*, A62:262–269, 2006.

[114] A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269, 1967.

[115] G. Wang and R. Dunbrack. PISCES: A protein sequence culling server. *Bioinformatics*, 19:1589–1591, 2003.

[116] Y. Weiss. Interpreting images by propagating Bayesian beliefs. In *Advances in Neural Information Processing Systems*, pages 908–915, 1996.

[117] Y. Weiss. Correctness of local probability propagation in graphical models with loops. *Neural Computation*, 12:1–41, 2000.

[118] Y. Weiss and W. Freeman. Correctness of belief propagation in Gaussian graphical models of arbitrary topology. *Neural Computation*, 13:2173–2200, 2001.

[119] E. Wigner. *Group Theory and its Application to the Quantum Mechanics of Atomic Spectra*. Academic Press, 1959. *Translated*: J. Griffin.

[120] J. Winn and C. Bishop. Variational message passing. *Journal of Machine Learning Research*, 6:661–694, 2005.

[121] J. Yedidia, W. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51:2282–2312, 2005.