



universität
wien

DIPLOMARBEIT

Titel der Diplomarbeit

Optimization Methods for Protein Folding

angestrebter akademischer Grad

Magistra der Naturwissenschaften (Mag. rer. nat.)

Verfasser:	Virginia M. Burger
Matrikel-Nummer:	0207332
Studienrichtung:	Mathematik
Betreuer:	Ao. Univ.-Prof. Dipl.-Ing. Dr. Hermann Schichl

Wien, am 25. 07. 2008

Contents

0	Introduction	1
1	Protein Structure Prediction	3
1.1	Proteins	3
1.1.1	Polypeptide Chains	3
1.1.2	Interactions between Amino Acids	4
1.1.3	Structure, States, and Function	5
1.1.4	Protein Data Banks	7
1.1.5	The Folding Process	7
1.2	Experimental Determination of Protein Structure	8
1.3	Prediction of Protein Structure: General Idea	9
1.3.1	Modeling the Problem	9
1.3.1.1	Minimizing an Energy Function	9
1.3.1.2	The Potential Energy Function	11
1.3.1.3	Difficulties and Solutions	13
1.3.2	Techniques	14
1.3.2.1	Template-Based Methods	14
1.3.2.2	Ab initio and De Novo methods	14
1.3.2.3	Fragment Assembly	15
1.3.2.4	Threading	15
1.3.2.5	Lattice Models	15
1.4	CASP	15
1.4.1	LGA	18
1.5	Progress in Protein Structure Prediction	20
1.5.1	Successful Strategies for Template - Free Prediction (at CASP)	20
1.5.2	Details of Improvements in LGA scores/RMSD over all CASPs, especially CASP7	23
2	Optimization	25
2.1	Introduction to Optimization	25
2.2	Local Optimization	27
2.2.1	Lagrangian, Optimality Conditions, and Duality	28
2.2.1.1	The Lagrangian	28
2.2.2	Optimality Conditions	31
2.2.2.1	Duality	32
2.2.3	Descent Methods	34
2.2.3.1	Line Search	34
2.2.3.2	Active Set Methods	35

2.2.3.3	Trust Regions and Newton-like Methods	36
2.2.4	Penalty methods and Interior Point/Barrier Methods	37
2.3	Discrete Optimization	37
2.3.1	Hidden Markov Models and Dynamical Programming	38
2.3.1.1	Probability of a Series of Observations $O = O_1, \dots, O_T$	40
2.3.1.2	Most likely hidden state sequence, given a series of observations	41
2.3.1.3	Training the Hidden Markov Model	43
2.3.1.4	Connection to Protein Folding	43
2.4	Global Optimization	44
3	Methods	45
3.1	Modeling the Protein's Geometry	45
3.2	Algorithm Summary	47
3.3	Phase I: Discrete Optimization	48
3.4	Phase II: Continuous Optimization	54
3.4.1	Description of the Variables and Parameters	55
3.4.2	Description of the Constraints	56
3.4.3	Optimization	58
3.5	Software	58
3.5.1	AMPL	58
3.5.2	Knitro	58
3.5.2.1	3 Algorithms	60
3.6	Implementation	60
3.6.1	Phase I: Discrete Optimization Program	60
3.6.2	Phase II: Continuous Optimization Program	62
3.6.3	Phase I and II Together	63
4	Results	65
4.1	The Original 27 Classes	66
4.2	HEFG classes	69
4.2.1	Comparison of Original Classes with HEFG classes	71
4.3	Secondary Structure Sequence Predictions	71
5	Conclusions and Outlook	75
5.0.1	Local or Global Optimization	75
5.0.2	Predictions with the HEFG Classification	75
5.0.3	Prediction of Secondary Structure Class Sequence	78
5.1	Proposed Phase III of the Algorithm	78
	Bibliography	78
	Resume	85

0 Introduction

This thesis focuses on the application of optimization to protein structure prediction. In Chapter 2, the difficulties of protein structure prediction are discussed, after a brief introduction to proteins in general. Then, current methods for protein structure prediction are introduced. In Chapter 3, some basic theory on optimization is explained, and optimization topics necessary to understand this paper are discussed. Chapter 4 explains the algorithm we used to try our hand at protein structure prediction. In Chapter 5 our results are discussed, which are at this point very inconclusive. In Chapter 6, we make conclusions and discuss possible ideas to continue work on this project.

1 Protein Structure Prediction

1.1 Proteins

Proteins are molecules in the cells of living beings. They perform and assist in many tasks in cells, such as catalyzing biochemical reactions, cell signaling, cell adhesion, and the cell cycle. They also have structural and mechanical functions. Proteins only function correctly when they are in their correct 3D shape. The goal of protein structure prediction is to predict the 3D shape of a protein based on its amino acid sequence. Prediction of structure seems necessary because the current experimental methods take around a month to determine one protein structure. There are less than 50,000 known protein structures in the Protein Data Base, but around 300,000 protein sequences in the SwissProt database, waiting to be determined [Craven, 2007]. An efficient and reliable method to determine the tertiary structure of a protein from its primary structure needs to be found in order to determine the structures of all the known protein sequences. Being able to predict protein structure would also allow bioinformaticians and biomedical engineers to design proteins that can accomplish specific tasks.

1.1.1 Polypeptide Chains

An amino acid is a molecule which contains an amino $-\text{NH}_2$ and a carboxyl $-\text{COOH}$ functional group. The 20 amino acids used in proteins were named the “magic 20” by Francis Crick and are the only amino acids coded for by the genetic code [Elliott and Elliott, 2002]. Nineteen of the twenty amino acids in proteins are α -amino acids, which have the form $\text{H}_2\text{N} - \text{C}^\alpha\text{H} - \text{COOH}$. The twentieth amino acid, proline, is actually an imino acid (contains an imino group, $-\text{NH}$ instead of $-\text{NH}_2$) and has a slightly different form. R, an organic substituent, is attached to the C^α atom of each amino acid and is unique to each amino acid. The structure of R varies greatly, and sometimes

includes heavy elements such as sulfur. The physical and chemical composition of each organic substituent R provides each amino acid with characteristic properties.

The availability of 20 amino acids with unique structures and properties allows evolution to create proteins with virtually every possible function. Amino acids with polar side chains are hydrophilic. That is, amino acids with an extra $-\text{NH}_2$ in the side chain are basic, and therefore hydrophilic, because they are positively charged. Analogously, amino acids with an extra $-\text{COOH}$ in the side chain are acidic, and also hydrophilic, because they are negatively charged. The nonpolar aliphatic amino acids, excluding Glycine, have hydrophobic side chains. Some amino acids are neither hydrophilic nor hydrophobic. For example, Glycine which has the most simple side group, $\text{R} = \text{H}$, is neutral to water. Amino acids with Sulfur in their side group, such as Cysteine, may have special purposes in protein structure. For example, they can form internal disulfide bonds or covalent S-S bonds, and they may provide external $-\text{SH}$ groups. [Elliott and Elliott, 2002]

A peptide bond forms between the carboxy terminal ($-\text{CO}$) of one amino acid and the amino terminal ($-\text{NH}$) of a neighboring amino acid by the removal of a water molecule. Two peptides bound by such a bond are called a dipeptide. Peptide bonds have a partial double bond character which limits rotation around the bond, making them very stable. Like a peptide, the dipeptide also has a carboxy terminal on one end and an amino terminal on the other, so it too can form peptide bonds. A small group of peptides bound by peptide bonds is called an oligopeptide. Peptides longer than around 80 amino acids are called polypeptides, and peptides with biological function are called proteins. Proteins usually have less than 2000 amino acids, but there are proteins with around 5000 amino acids. Amino acids in proteins are often referred to as residues. [Elliott and Elliott, 2002]

1.1.2 Interactions between Amino Acids

In addition to the peptide bonds, there are many non-bonded interactions in proteins. Hydrogen bonds form between two negatively charged atoms when they share a positively charged hydrogen atom. In proteins, these bonds occur between atoms on different side chains, atoms on side chains and atoms in the solvent, atoms on side chains and atoms on the polypeptide backbone (the protein's $-\text{CH} - \text{CO} - \text{NH} - \text{CH} - \text{CO} - \text{NH} - \text{CH} -$ chain), atoms on the polypeptide backbone and atoms in the solvent,

and between atoms on the polypeptide backbone in different amino acids. Hydrogen bonds are often very important elements of the protein's tertiary structure. Hydrogen bonding is the driving force of the formation of alpha helices and beta sheets, which will be discussed in the next section.

Hydrophobic bonds occur between hydrophobic side chains, in order to prevent hydrophobic amino acids from coming into contact with water molecules. These bonds play a major role in the packing of hydrophobic side chains in the protein core, when the protein is in a hydrolic solvent such as water.

Van der Waals forces are weak electrostatic forces causing attraction and repulsion of atoms. Because of the compact packing of atoms inside a protein, most atoms inside of a protein are involved in some kind of Van der Waals interaction with another atom in that protein. Disulfide bridges can occur between two Cysteine molecules. These, like peptide bonds, are covalent bonds, and thus very strong.

1.1.3 Structure, States, and Function

A protein's **primary structure** is its amino acid chain. The **tertiary structure** is the protein's 3D shape. Proteins also have a **secondary structure**, which describes local interactions between the amino acids.

Specifically, secondary structure describes the hydrogen bonds between the amide groups. The C = O and the N - H groups on the polypeptide backbone are polar and capable of forming hydrogen bonds. The protein structure will be instable if these groups are not able to engage in bonds and release free energy. Thus, hydrogen bonds form between the carboxyl group of one amino acid with the amino group of another amino acid inside the protein. These bonds occur in several common patterns.

In *alpha helices*, the C = O of each peptide bond forms a bond with the N - H group of the amino acid four residues ahead. This forms a right-handed helix. The side groups project outside of the helix, and the helix is so tightly coiled that it is virtually solid. The Van der Waals radii of the atoms in the helix allow almost no movement, making the helix very stable. Alpha helices can be very long, but average ten amino acid residues in length. Some amino acids are more likely to form alpha helices than others. If a proline occurs in a helix, the helix immediately breaks, because the imino

group of the proline residues lacks a spare hydrogen atom for bonding. [Elliott and Elliott, 2002]

Beta strands occur between two adjacent amino acid chains. Neighboring chains are either formed by two adjacent polypeptide chains or by one polypeptide chain folding back upon itself. Similar to alpha helices, each hydrogen bond occurs between the carboxyl group of one chain and the amino group of the other chain. These formations often occur between several rows of neighboring amino acid chains, forming “sheets”. If the polypeptide chains run in the same direction, the β -sheet is termed parallel, and if they run in opposite directions, the β -sheet is termed antiparallel. The neighboring C_α atoms on adjacent sheets are alternatively slightly above and then below the plan, giving the β -sheets a pleated appearance. These formations are, like α -helices, very stable.

Both α -helices and β -sheets can occur in the same protein. The tertiary structure is the entire three dimensional structure, so it describes the size, orientation and positioning of the α -helices and β -sheets, as well as the *connecting loops*, the regions between them. The structure of the connecting loops (which are also referred to by the misnomer “random coils”) is largely determined by side chain interactions. Connecting loops are often found on the exterior of proteins, since they usually can not form hydrogen bonds with the carboxyl or amino groups, nor with other side chains. The tertiary structure is compact and, when in hydrolic solvent, hides all of the hydrophobic side chains on the inside of the protein. This minimizes interactions between water and hydrophobic groups, and maximizes the number of van der Waals bonds between hydrophobic groups. Membrane proteins may have a different distribution of hydrophilic and hydrophobic groups, depending on their location in the cell membrane’s tri-lipid bilayer.

A protein’s function depends on its structure. Proteins have a region on their surface called a *binding site* which allows them to bind specifically and tightly to another molecule. This region is key in allowing a protein to perform its function. The binding site is defined by the protein’s tertiary structure through its shape and through the properties of the amino acids surrounding it. Some proteins do not function alone, but in a single functional complex. Each protein in the complex is called a subunit, and the complex itself is called a multi-subunit protein. The structure of the protein complex is referred to as the *quaternary structure*.

A protein can only function when in its native, folded state. Heat, change in pH value, and other chemical disturbances can cause a protein to lose its correct tertiary structure and unfold. The protein then becomes a randomly configured polypeptide chain, which can not function properly because its binding site is no longer intact.

1.1.4 Protein Data Banks

There are various data banks which store information about known protein structures. The PDB (Protein Data Bank) provides cartesian coordinates (found with NMR-Spectroscopy or X-ray Crystallography) for all atoms of proteins and nucleic acids. It uses an alphabet of 25 numbers to label the amino acids. 20 of the numbers correspond to specific amino acids, and the other five numbers have different meanings. They represent, for example, the end of a chain or an indistinguishable amino acid.

The DSSP (Definitions of the Secondary Structures of Proteins Database) contains a selection of high-quality data from the PDB. It provides coordinates of the C_{α} -atoms of a protein and gives information about the protein's secondary structure.

1.1.5 The Folding Process

The protein folding process begins in the cell's nucleus, when a protein complex called RNA polymerase transcribes a gene onto a messenger RNA. Transcription is the process of transferring the gene's DNA sequence onto the RNA base sequence. The messenger RNA then leaves the nucleus, where it ribosomes attach themselves to it. The ribosomes then translate the RNA sequence into the amino acid sequence using the genetic code. Each combination of three nucleotides, called a codon, codes for one amino acid. Usually a few different combinations of three nucleotides code for the same amino acid, which improves the amino acid sequence's resistance to mutation. After being synthesized, the protein folds into its native state in a time scale between a few microseconds to a few minutes. Sometimes, special molecules called "chaperones" help the protein fold by providing conditions under which the protein can fold properly. For example, they might bind temporarily to hydrophobic chains, in order to prevent such chains from binding improperly. These molecules direct the folding process. Once the protein is in the proper conditions for folding, the folding process depends solely on the amino acid sequence and the surrounding solvent. [Elliott and

Elliott, 2002]

1.2 Experimental Determination of Protein Structure

X-ray Crystallography and NMR-Spectroscopy are the two main methods used to determine a protein's folded structure.

X-ray Crystallography produces a diffraction pattern specific to a protein's structure by focusing X-ray beams through a crystal of the protein. The protein structure is then determined through analysis of the diffraction pattern and information previously gathered about the protein. Because proteins are difficult to purify and easily unravel from their 3D structure, it is quite difficult to crystallize a protein. Many proteins can not be crystallized at all, and, even if a protein is able to be crystallized, it can take many months to create an acceptable crystal form. The analysis of the diffraction pattern can usually be completed in a day's time, however. Larger proteins are more difficult to analyze than smaller proteins, and their analysis usually requires a complicated process called Multiple Isomorphous Replacement.

NMR-Spectroscopy probes the magnetic environment surrounding the nuclei of the protein's many atoms by applying magnetic fields and high-frequency radiation to the protein. It is much easier to create a protein sample for NMR than for Crystallography, since the protein must only be purified, as opposed to being crystallized. However, processing the data returned from spectroscopy is a long, tedious process, usually involving a global optimization problem.

There are thousands of proteins whose tertiary structures need to be determined, and both X-ray crystallography and NMR-spectroscopy require several months to determine a single protein's tertiary structure. At that rate, it could take centuries before all of the structures are determined, which is definitely too long. If a protein is too large, many additional tasks must be performed in order to determine the structure. Although these methods provide extremely accurate results, to be really helpful in nanotechnology and medical sciences, they would need to be able to determine any protein's structure within a much shorter period of time.

1.3 Prediction of Protein Structure: General Idea

The general idea behind protein structure prediction is to choose an energy function which describes the protein's tertiary structure, and then minimize this function. The global minimum of the ideal energy function is then, theoretically, the protein's correct structure.

1.3.1 Modeling the Problem

The basic idea behind most models for protein structure is that proteins are in their native state when their potential energy is minimal. First, the reasons for this assumption will be explained, and then finding a function to describe the potential energy of a protein will be discussed. This section is based on Neumaier [1997].

1.3.1.1 Minimizing an Energy Function

First, we show that as the temperature T approaches zero and the time t approaches infinity, a protein continues losing energy until it comes to a rest at a stationary point of its potential:

Proof Idea: Start with the Langevin Dynamics Equation:

$$M\ddot{x} + C\dot{x} + \nabla V(x) = D\dot{W}(t)$$

The mass matrix M is diagonal, where m_{ij} corresponds to the mass of the i^{th} atom in each of its three dimensions, $j = 1, 2, 3$. The damping matrix, C , is a positive definite scalar multiple of M . The coupling of the protein to the environment is represented by C and D . Together, D , the diffusion matrix, and W , a Wiener Process, represent normalized white noise.

The above equation depends solely on the covariance of the noise:

$$DD^T = \langle \xi \xi^T dt \rangle$$

The Fluctuation-Dissipation Theorem says that

$$DD^T = 2k_B \cdot T \cdot C.$$

[Neumaier, 1997] If we set $C = \gamma \cdot M$, then

$$DD^T = 2k_B \cdot T \cdot \gamma \cdot M.$$

Taking the square root implies that

$$D = \sqrt{2k_B \cdot T \cdot \gamma} \cdot M^{\frac{1}{2}}.$$

If we then send the temperature to zeros, then there can be no random forces, which means that

$$\langle \xi \xi^T dt \rangle = 0.$$

Since $DD^T = \langle \xi \xi^T dt \rangle$, we then have $DD^T = 0$, which means that $D = 0$. Thus, the Langevin Dynamics Equation is reduced to

$$M\ddot{x} + C\dot{x} + \nabla V(x) = 0, \tag{1.1}$$

and is now an ordinary differential equation for x .

It is a well-known fact that energy is the sum of kinetic and potential energy: $E = E_K + E_P = \frac{1}{2}\dot{x}^T M \dot{x} + V(x)$. If we take the derivative of this, we have:

$$\begin{aligned} \dot{E} &= \dot{x}^T M \dot{x} + \nabla V(x)^T \dot{x} \\ &= (\dot{x}^T M + \nabla V(x)^T) \dot{x} \\ &= ((M^T \dot{x})^T + \nabla V(x)^T) \dot{x} \\ &= (M^T \dot{x} + \nabla V(x))^T \dot{x} && | (1.1) \\ &= (-C\dot{x})^T \dot{x} && | C \text{ is positive definite} \\ &= -\dot{x}^T C \dot{x} < 0 \end{aligned}$$

Therefore, the derivative of the energy is negative, which means that it is decreasing. If $\dot{x} = 0$, that is, if there is no movement, since $\dot{x}$ refers to the velocity, then all energy is potential energy: $M\ddot{x} + C\dot{x} + \nabla V(x) = \nabla V(x) = 0$

The stationary point of $V(x)$ is usually a local minimum. The protein's potential energy surface can be very complicated and contain thousands of local minima. Random forces allow the protein to move between different metastable states, which are local minima of the energy function.

We now show that the protein prefers the metastable state where the potential has its

global minimum.

Proof Idea: The Arrhenius Law says that the mean transition frequency f between two different energy states with energy difference equal to δE is

$$f = \frac{k_B T}{h} e^{\left(\frac{-\delta E}{k_B T}\right)},$$

where h is Planck's constant. Consider two adjacent local minima with potentials E_1 and E_2 , ($E_1 < E_2$), and transition state E_{12} , that is E_{12} is the peak of the curve between the two minima. Define the transition energies as

$$E_{1 \rightarrow 2} := E_{12} - E_1 \quad E_{2 \rightarrow 1} := E_{12} - E_2.$$

Let $f_{1 \rightarrow 2}$ be the mean frequency of transition from E_1 to the transition state E_{12} , and $f_{2 \rightarrow 1}$ be the mean frequency of transition from E_2 to the transition state E_{12} . We show that the mean frequency of transitions away from E_1 is less than the mean frequency of transitions away from E_2 .

$$\begin{aligned} \frac{f_{1 \rightarrow 2}}{f_{2 \rightarrow 1}} &= \frac{\frac{k_B T}{h} e^{\left(\frac{-E_{1 \rightarrow 2}}{k_B T}\right)}}{\frac{k_B T}{h} e^{\left(\frac{-E_{2 \rightarrow 1}}{k_B T}\right)}} \\ &= e^{\left(\frac{-(E_{1 \rightarrow 2} - E_1)}{k_B T} - \frac{-(E_{1 \rightarrow 2} - E_2)}{k_B T}\right)} \\ &= e^{\left(\frac{E_1 - E_2}{k_B T}\right)} = 1/e^{\left(\frac{E_2 - E_1}{k_B T}\right)} < 1 \\ &\Rightarrow f_{1 \rightarrow 2} < f_{2 \rightarrow 1} \end{aligned}$$

Therefore, it can be assumed that a protein's native configuration is the configuration which has the smallest potential energy.

1.3.1.2 The Potential Energy Function

Many predictors try to find an energy function which describes the protein's energy surface and then minimize it because of the assumption above that the configuration with the smallest potential energy is the most likely native structure of a protein. One way of creating an energy function to describe the potential energy of a protein's configuration is by using molecular mechanics. In molecular mechanics, physics-based energy functions incorporate quantum mechanics into a molecular mechanical force

field:

$$E = E_{\text{Bonded Interactions}} + E_{\text{Non-Bonded Interactions}}$$

Non-bonded interactions include Van der Waals attractions, electrostatic repulsion/attraction, etc.

The bonded terms account for distortion from the ideal values of the bonds, angles, etc., which have somehow (usually experimentally) been determined. Most of the terms are based on Hooke's Law: $F = -kx$, where F is the force exerted, x is the distance between new value and equilibrium, and k is a constant.

Knowledge-based energy functions use information from known protein structures to develop empirical potentials:

$$E = E_{\text{generic}} + E_{\text{secondary}} + E_{\text{tertiary}}$$

E_{generic} refers to qualities typical of all proteins, and $E_{\text{secondary}}$ (resp. E_{tertiary}) refer to qualities specific to a particular sequence.

Many energy functions use a mixture of physics-based and knowledge-based elements. While defining an energy function, it is important to consider smoothness and shape of the function, the number of variables, and the calculation time, because finding the minimum of the function must be feasible in a reasonable amount of time.

An example of an empirical physics-based energy equation, which is used in many prediction algorithms, is:

$$E_{\text{AMBER}} = \sum_{\text{bonds}} K_{r_i} (r_i - r_{i,\text{eq}})^2 + \sum_{\text{angles}} K_{\theta_l} (\theta_l - \theta_{l,\text{eq}})^2 + \sum_{\text{dihedrals}} \frac{V_k}{2} (1 + \cos(n_k \phi_k - \gamma_k)) + \sum_{i < j} (\varepsilon_{ij} ((\frac{\sigma_{ij}}{r_{ij}})^{12} - 2(\frac{\sigma_{ij}}{r_{ij}})^6) + C(\frac{q_i q_j}{r_{ij}}))$$

The distance between the i^{th} atom and the previous atom along the atom chain is given by r_i , and the distance between two atoms i and j is given by r_{ij} . The charge of the i^{th} atom is given by q_i . Dihedral angles formed by four bonded atoms are described by ϕ_k , and θ_l is the bond angle formed by three bonded atoms.

There are often more terms included for knowledge-based energies, hydrogen bonds,

out of plane bending, and solvent (implicit vs. explicit). The terms must be given weights, which are usually determined heuristically or through some kind of knowledge, for example through secondary structure predictions.

1.3.1.3 Difficulties and Solutions

Protein structure prediction is difficult for many reasons. Just finding the global minimum of an energy function is difficult, because global optimization is hard in the first place, and is extremely hard when there is a large search space, a large number of variables, and many local minima.

To solve the global optimization problem efficiently, some simplifications must be made. Many methods simplify the problem drastically at the beginning of the minimizing process, in order to filter out unlikely tertiary configurations, and then expand the problem back into its original form, in order to find the global minimum of the reduced problem.

Many methods represent the protein in torsional space by their torsional angles, instead of cartesian space, which reduces the size of the problem threefold. It is also common to ignore all or most of the detail of the protein's side chains. United-atom approaches represent some of the carbon and hydrogen atoms as single, combined units.

Thanks to new methods in parallel computing, and increasingly powerful supercomputers, it is becoming possible to perform increasingly detailed searches in a reasonable amount of time. The Folding@HOME program works by using distributed computing to sample an extremely large set of protein structures. Subscribers to Folding@HOME allow their computers to be used to sample different structures while they are idling. [Pande and University]. This program has had great success, and many other prediction groups have designed similar programs, such as Rosetta@HOME by the Baker Lab. Recently, David Baker created a protein folding game, which lets people fold for fun. [gam]

1.3.2 Techniques

1.3.2.1 Template-Based Methods

Template-based methods assume that proteins with homologous sequences have similar structures. They use the Protein Data Base to find template sequences homologous to the new sequence, and base the structure of the new sequence on the structures of the template sequences. This prediction method can be very reliable for proteins, if homologous sequences can be found. In 2003, 90% of predictions, in cases where a sequence could be found which was at least 65% identical to the new sequence, were accurate within 5Å, and 65% were accurate within 2Å. Template-based methods require the existence of a homologous sequence, which can not always be found. For protein regions where no suitable template can be found, de novo methods must be used.

1.3.2.2 Ab initio and De Novo methods

De novo methods predict the tertiary structure of proteins using no, or very little, information from homologous structures. Methods using only physics to predict the protein structure are termed ab initio methods. Since it is assumed that the protein's native structure is the configuration with the lowest potential energy of all possible configurations, a search algorithm is typically used to find the global minimum of the potential energy function. Very good predictions using these methods are within approximately 5Å of the actual structure.

De novo methods are much more computationally demanding than template-based methods, and they do not usually provide the same degree of accuracy. Template-based predictions can be completed in seconds, whereas de novo predictions can take days. Template-based methods can be used for much larger proteins and are more accurate than de novo based methods. However, in many cases, there is no suitable template for the protein, or for regions of the protein, and de novo structures must be used. By predicting protein structures for nonhomologous sequences through de novo methods, the selection of protein structures provided by the Protein Data Base becomes more diverse, allowing more template-based predictions.

1.3.2.3 Fragment Assembly

Many de novo methods use fragment assembly techniques because they implicitly bias the search to configurations with good local structure. That is, they prevent clashes and promote good local packing because they contain knowledge-based information. The fragment assembly method works by first creating a library of small (3–10 amino acids) fragments of protein sequence with known structure. These fragments come from many different proteins. Then, the configurations are minimized by substituting fragments into the target protein and evaluating a scoring function for each substitution. Heuristic techniques, such as Monte Carlo, are usually implemented to prevent the algorithm from getting stuck in local minima of the scoring function. This method is used by both Rosetta and TASSER, two of the best groups in template-free prediction at recent CASPs. (See 1.4).

1.3.2.4 Threading

Threading methods try out many different possible structures for a protein sequence. A scoring function, such as a potential function, evaluates the correctness of each structure for the target sequence.

1.3.2.5 Lattice Models

Lattice methods are used to efficiently search through possible configurations. By digitizing the possible configurations onto a grid, and representing each amino acid by only a few atoms, they are able to exhaustively search the configuration space of small proteins. For larger proteins, however, heuristic search methods must be used in order to search through the possible configurations. Lattice models are used by TASSER for structure assembly simulation, see 1.4.

1.4 CASP

The Critical Assessment of Techniques for Protein Structure Prediction (CASP) is a biannual community-wide experiment, with the goal of assessing the current ca-

pabilities in the area of protein structure prediction. Around April, CASP receives information from scientists about proteins whose tertiary structures will be determined shortly, but are not yet known or available to the public. The primary structures of these target proteins are then made available to predictors, so that they can perform a true prediction of their structures. This allows the community of researchers interested in protein structure prediction to get a grasp of the current standards and the best new ideas every other year.

In CASP7, most of the target proteins came from structural genomics projects. A formalized procedure for the transfer of information between the structural genomics centers, the PDB, and the Prediction Center was developed for the two largest target contributors. After determining each target's structure, the structural genomics centers provided the Prediction Center with each primary structure, and withheld each experimentally-derived tertiary structure from the PDB for about three weeks, so that the predictors had three weeks to predict each structure. Having a pre-determined tertiary structure eliminated the past problem of targets being lost because their structure could not be successfully experimentally determined or were not applicable for prediction.

Some of the goals of CASP8 are to see how similar the models are to the experimental structure, to see if there has been progress since 2006's CASP7, and to see what methods of protein structure prediction are most effective [Center, 2008]. CASP is also interested in finding out what the current difficulties in protein structure prediction are [Moult et al., 2007].

The targets in CASP are divided into three broad categories based on their difficulty level in respect to similarity to a template. Originally, the comparative modeling category was for targets for which a template or templates, covering most of the target, could easily be found with a database search such as BLAST (the basic alignment search tool, see bla). The fold recognition category was for targets for which a template could be found, but with more complicated methods than a BLAST search. The free modeling category covered targets for which no suitable template could be found, or for which a template-based prediction was not expected to be fruitful. Over the first few CASP competitions, template-based predictions improved greatly, and in CASP7, the comparative modeling and fold recognition categories were merged into one template-based modeling category, and a new high resolution category was defined. Targets in the high resolution category are expected to be predicted accurately enough

that side chains, loops, and active sites can also be considered. The accuracy of the backbone prediction for these models is expected to rival the accuracy of experimental structures. The free modeling category remained the same, but its name was changed to template-free modeling. [Moult et al., 2007] Categories also exist for the prediction of locations of binding sites, and for predicting the quality of the model.

The correct evaluation of predictions is a topic closely related to CASP. Independent assessors, as in assessors who are not related to the Prediction Center, the experimentalists, or the predictors, evaluate the predictions. Each category of prediction has its own class of assessors. The Prediction Center provides the structure prediction assessment teams with numerical data from the two main metrics, GDT_TS and ALO, as well as from other structure comparison software to help them evaluate the predictions. The assessors may also use their own metrics and methods to help their evaluation. The GDT_TS and GDT_HA algorithms are described in the next section. ALO is a metric which measures alignment accuracy. It reflects how well a model can be superimposed on its target. [Moult et al., 2007]

In CASP7, GDT_TS was used as the main evaluator for template based and template free models. Because some of the template based models were of such a high accuracy in CASP7, GDT_HA was used in addition to GDT_TS to provide a finer measure of accuracy for high resolution models. Visual inspection was also found to be important for template free models, which are typically much less accurate than template based predictions. Visual inspection and a metric are used together for evaluation of template free models, because the evaluation of template free models is very subjective. That is, if a model varies significantly from its target, one assessor may put more weight on the overall appearance of the model in comparison to the target, while another assessor may think it is more important that the model has a large domain where the local structure is very similar to that of the target.

Jauch et al. [2007] describes the visual evaluation performed by their group: Each target was evaluated separately. First, the 50 best models of the target, based on GDT_TS, and CMO, another evaluation algorithm developed by the assessment team, were chosen. Then, each of the three assessors was given the 50 best models, as well as 25 other models randomly chosen specifically for him. Independently, each assessor chose five models as the best predictions of that target. The three groups of five models were then combined and sent back to the assessors. The assessors then independently chose three models from the combined group. Finally, all of the

assessors met up to discuss their choices, and each assessor was allowed to redraw one of his selections, if he decided after the discussion that it was less accurate than the other models. Typically, models that were highly ranked by GDT_TS were also highly ranked by visual inspection, but many models that were ranked highly by visual inspection did not rank highly by GDT_TS. GDT_TS seems to rank models with accurate local structure highly, while visual inspectors tend to rank models with more accurate topology highly [Jauch et al., 2007]. In CASP8, similar evaluation methods will be used [Center, 2008].

1.4.1 LGA

GDT_TS is a part of the Local-Global Alignment method, which was designed to compare protein structures with respect to both local and global structure superpositions [Zemla, 2003]. LGA was designed by Zemla [2003] for both sequence-dependent and sequence-independent comparisons, that is, for comparisons with or without a preassigned residue-correspondence. This project only makes use of the sequence-dependent comparison, so only that aspect of LGA will be described here. Two components are used in LGA’s scoring function, the “longest continuous segments” (LGA) and the “global distance test” (GDT).

To understand LGA, the term root mean squared deviation must be explained. Root mean square deviation (RMSD) is a common way of evaluating the deviation between two objects. For two vectors, $x^m = (x_1^m, \dots, x_n^m)$ and $x^t = (x_1^t, \dots, x_n^t)$, the root mean squared deviation is given as:

$$\text{RMSD}(x^m, x^t) = \sqrt{\frac{\sum_{i=1}^n \text{error}(x_i^m, x_i^t)}{n}},$$

where the error function is typically the squared distance between the two points, for example $\|x_i^m - x_i^t\|^2$. Before calculating this, one of the vectors must be superimposed onto the other, with the goal of finding a minimal RMSD. Here, RMSD is used to find the deviation of a model for a protein’s structure from its template, the actual protein structure. Therefore, x^m is a vector with the coordinates of the consecutive atoms in the model structure, and x^t is a vector with the coordinates of the corresponding atoms in the template structure.

If the model has some incorrect bond angles, but is a perfect replicate of its target in between those angles, that is, it has all of the correct structural elements, but they

are not aligned properly, then its RMSD will not take the correct areas into account, but will only reflect that the RMSD for the best superposition of the target onto the model is much higher than it would be for a perfect prediction. CASP assessors try to find more suitable methods of evaluating predictions that take into account both the overall structure and the accuracy of local regions. [Jauch et al., 2007]

LCA searches for the longest continuous segment of residues that can fit under a given C_α RMSD cutoff when one protein is superimposed onto the other. [Zemla, 2008] LCA tests many local superimpositions and searches for continuous segments under an RMSD of 1Å, 2Å, and 5Å. Using several C_α RMSD cutoffs allows LCA to collect more information on similarities between the two protein structures.

GDT finds all of the residue pairs within a given distance cutoff when one protein is superimposed onto another. The current procedure is described in detail by Zemla [2008] and Zemla [2003]: Many initial superpositions of the target onto the model are calculated, using each three-residue segment and each continuous segment found by LCS as a starting point. The number of equivalent residue pairs, that is model-target C_α pairs under a given distance cutoff is counted, and then iteratively extended. The iteration is as follows: First, the superposition is performed and the largest set of residue pairs under the distance cutoff is found. Then, the residue pairs that are not under the distance cutoff are removed from the set, and the superposition is performed again. The iteration is repeated until the set of residues pairs under the distance cutoff remains the same for two cycles.

The two components complement each other in that LCA finds areas of local similarity between proteins, while GDT gives information about how similar the overall structures are. The LGA score is given by Zemla [2008]:

$$LGA_S = \omega * S(GDT) + (1 - \omega) * S(LCS).$$

The weight variable ω is between 0 and 1. The function S is defined for GDT as

$$S(GDT) = 2 * \frac{k * GDT_{v_1} + (k - 1) * GDT_{v_2} + \dots + 1 * GDT_{v_k}}{(k + 1) * k},$$

where GDT_{v_i} estimates the percent of residues in the target molecule that are within v_i Å of the corresponding residue in the model, when the target is optimally superimposed onto the model. In this case, $v_i = 0.5, 1.0, 2.0, 4.0, \dots$

For LCS, S is defined as

$$S(LCS) = 2 * \frac{k * LCS_{v_1} + (k - 1) * LCS_{v_2} + \dots + 1 * LCS_{v_k}}{(k + 1) * k},$$

where LCS_{v_i} is the percent of residues in the target molecule (after being optimally superimposed onto the model), for which the root mean squared deviation from that residue to the corresponding residue in the model is less than $v_i \text{\AA}$, for $v_i = 1.0, 2.0, 4.0, 8.0, \dots$

GDT_TS and GDT_HA are two versions of GDT where only four distance cutoffs are used to calculate $S(GDT)$. The cutoffs for GDT_TS are 1.0, 2.0, 4.0, and 8.0Å and the cutoffs for GDT_HA are 0.5, 1.0, 2.0, and 4.0Å.

GDT_TS scores range from 0 to 100, 100 being the highest possible score. RMSD scores range from zero to infinity, zero being perfect, and higher scores being increasing bad. However, a score of infinity is virtually impossible to obtain when comparing protein structures.

1.5 Progress in Protein Structure Prediction

1.5.1 Successful Strategies for Template - Free Prediction (at CASP)

The Baker Group with Rosetta@HOME and the Zhang group were the most successful groups at CASP7 in the template-free prediction category. Both of these groups had several prediction strategies for different prediction categories. Declaring one group the best is hard, because some groups participated in many more target predictions than others, making it difficult to compare overall results.

Rosetta@HOME performed best, in comparison to competitors, in the template-free group and in the template-based category for proteins with less than 200 residues and no template with more than 30% sequence identity. According to Jauch et al. [2007], Baker performed best by visual inspection, receiving 18 of 54 possible votes, while Zhang only received 6 votes. By GDT_TS scores, Baker and Zhang were both the best and could not be distinguished from each other, but Baker was significantly better than some groups that were not distinguishably better or worse than Zhang. Therefore, the Baker group was the best by both criteria for template-free predictions. The Zhang

group performed better than the Baker group in template-based predictions of longer proteins with greater than 20% sequence identity with a template of known structure and in predictions of proteins with greater than 30% sequence identity with a template of known structure. [Das et al., 2007]

The Baker Group had two separate groups competing in the CASP7 competition: Robetta Server and Rosetta@HOME. For template-free predictions, both groups used the same fragment assembly strategy to generate a large pool of conformations, but Rosetta@HOME then ran a full-chain, all-atom refinement on each conformation. Rosetta@HOME was able to run such an extensive computation on each conformation because of its enormous computational power. An average of 500,000 CPU hours were available for each target through distributed computing. Rosetta@HOME was more successful than Robetta Server for template-based predictions, but both groups were ranked in the top 20 by GDT_TS scores and were chosen at least fivetimes by visual inspection to be one of the three best. [Jauch et al., 2007] Das et al. [2007] concludes that the dominance of Rosetta@HOME over Robetta server shows that the all-atom refinement really improves predictions, since all-atom refinement was the only real difference between the two prediction methods.

Rosetta@HOME uses an all-atom energy function and aggressive sampling. The energy function is physically realistic. It contains terms for van der Waals interactions and free energy of solvation, and uses an orientation-dependent hydrogen bonding potential. For template-free predictions in CASP7, it first chose a diverse set of fragments to match secondary structure predictions made with secondary structure prediction programs. Then it used Rosetta fragment assembly with a low resolution energy function to find a large set of possible conformations. The energy function used favored hydrophobic burial and β -strand packing. Then, with bond lengths and angles fixed at ideal values, it performed a full-chain, all-atom refinement. With this method, $10^5 - 10^6$ conformations were found for each target. Using clustering and human evaluation, 5 of the best 100 - 1000 were chosen as models for the CASP competition. [Das et al., 2007]

One of the benefits of an all-atom energy function is that it creates physical features such as hydrogen bonds with native like geometry automatically. The largest drawback is obviously the enormous computing power required. Due to computational limitations, the Baker Group was not able to perform all-atom refinement on proteins longer than around 200 residues. It seemed to perform the best where it could

use several iterations of all-atom refinement, though. For CASP8, Rosetta@HOME wants to start using a combination of several templates to make conformations for its template-based predictions. The Zhang group used much more evolutionary knowledge in its template alignment, and had better results in the template-based category where there was no one clear template. Rosetta@HOME also wants to take into account the target protein's interactions with its neighbors, because that has an effect on the overall free energy. [Das et al., 2007]

The following section is based on Zhang [2007]. The Zhang Lab considers itself the most successful group at the CASP7 competition. Their algorithm is called TASSER, meaning "Threading, Assembly, Refinement." In CASP7, they used a modified version of TASSER called I-TASSER. The Zhang group competed as a human group, "Zhang", and as a server only group, "Zhang Server," in CASP7. The human group was slightly better than the server group, especially in the free-modeling category. Both groups used essentially the same system to predict protein structure, but there were some differences.

Zhang wrote "The protein structure prediction problem could be solved using the current PDB library" and his extensive sequence alignment and threading efforts show that he believes his hypothesis. The I-TASSER algorithm uses a wide variety of techniques to produce excellent predictions: dynamic programming, lattice models, threading, alignment, a very complicated force field, modified replica-exchange Monte Carlo simulation, neural-networks, clustering, and domain separation. Dynamical Programming is used to find the best template sequences for threading. The force field includes many knowledge-based energy terms. These include terms for short-range C_α correlations, side chain interactions, and C_α distance constraints. Terms dependent on secondary structure predictions (made by prediction software like with Baker) are also included, such as terms related to hydrogen bonding between backbone atoms. This potential is then used to assemble structures (structures built from threaded fragments) on a lattice with Monte Carlo simulation. Afterwards, the structures are refined. I-TASSER only predicts C_α atoms. It uses Pulchra [Rotkiewicz and Skolnick, 2008] to add the backbone atoms, and Scwrl [Canutescu et al., 2003] to build the side chains.

The human group used visual inspection to separate protein domains, whereas the server group separated the domains computationally, and that seems to be one of the reasons that the human group performed better in template-free modeling.

The Zhang group is working on improving long-range contact predictions for the template-free models, as well as developing a version including more atoms, so that it can make more accurate side-chain and hydrogen-bonding predictions.

1.5.2 Details of Improvements in LGA scores/RMSD over all CASPs, especially CASP7

The main criteria for measuring progress between CASPs is comparison of the average value (the value depending on whichever scoring function was used in both CASPs being compared) of the six best models of a target from one CASP competition and the average value of the six best models of a target of similar difficulty from a different CASP competition. The best GDT_TS score for targets from one CASP competition are also compared with the best GDT_TS score from targets of a similar difficulty level in a different CASP competition. Difficulty is determined by the availability of a protein in the PDB with similar sequence and the similarity of the structure of the target protein to proteins of known structure which could be used as templates. In all CASP competitions, there have been targets from a large range of difficulties. The largest improvement between CASP6 and CASP7 was in server predictions. Servers were originally only used to generate starting models for the human predictors, but in CASP6 began making predictions comparable to those made by humans, and in CASP7 server predictions were actually better than human predictions on a few occasions. [Kryshtafovych et al., 2007]

Template-free predictions did not improve significantly between CASP6 and CASP7. In CASP7, the median GDT_TS score for proteins of length 50 was about 32.5, and in CASP6 about 30.5. For proteins with around 130 residues, the interpolated median GDT_TS score improved from 20 to 22.5 between CASP6 and CASP7 (See figure 5.1). Thus, for larger proteins, there was a slight improvement in the median score. However, the targets with better (relative to CASP6) scores in CASP7 were on the borderline of being template-based, and if these targets were removed from the analysis, there would be no significant improvement. The best GDT_TS score for each target did not improve between CASP6 and CASP7. [Jauch et al., 2007]

2 Optimization

2.1 Introduction to Optimization

Most of the information in this chapter comes from Professor Hermann Schichl's Optimization and Variational Calculus course from winter, 2006, and from Neumaier [2007]. Otherwise, sources are given in the text.

An **optimization problem** is a problem of the form:

$$\begin{aligned} \min_x f(x) \\ \text{s.t. } x \in \mathcal{F}, \end{aligned}$$

where $\mathcal{F}$ is the **feasible set**, $f : D \rightarrow X$, and $x \in D$. The set D can be anything, but is usually a subset of $\mathbb{R}^n$. The set X is any totally ordered set (needed to easily define $\leq$) and is usually $\mathbb{R}$. The goal is to find a value $\hat{x}$ which is in the set $\mathcal{F}$, so that there is an $\varepsilon > 0$ such that $f(\hat{x}) \leq f(x)$ for all $x \in B_\varepsilon(\hat{x}) \cap \mathcal{F}$. If such an $\hat{x}$ can be found, then it is called a **local optimum** of f , and the problem has been solved. By setting $f = -f$, the maximum can also be found. If the goal of the problem is to find an $\hat{x} \in \mathcal{F}$ so that $f(\hat{x}) \leq f(x)$ for all $x \in \mathcal{F}$ (the **global minimum** of f) then the problem is called a **global optimization problem**.

The feasible set $\mathcal{F} = \{x \in D \subseteq \mathbb{R}^n \mid F(x) \leq 0, H(x) = 0, x \in B\}$ is made up of the inequality constraints $F : D \rightarrow \mathbb{R}^k$, the equality constraints $H : D \rightarrow \mathbb{R}^m$, and a set $B \subseteq D$. The set B has the form $B = \prod_{i=1}^n B_i$, where each B_i is a simple constraint on the component x_i of x . The constraint may be a simple bound (i.e. $x_i \in [a_i, b_i]$) or an integer constraint (i.e. $x \in \mathbb{Z} \cap [a_i, b_i]$) or a binary constraint (i.e. $x \in \{0, 1\}$), etc.

Writing out the feasible set, the above problem is equivalent to:

$$\begin{aligned} & \min_x f(x) \\ & s.t. \ F(x) \leq 0 \\ & \quad H(x) = 0 \\ & \quad x \in B. \end{aligned}$$

The structure of $\mathcal{F}$ and the objective function f determine the type of optimization problem. Some common types of optimization problems are:

- **Linear Program:** The functions f , F , and H are all linear, and there are no integer constraints. The set of simple constraints on the variables, B , must be of the form $B = \prod_{i=1}^n [a_i, b_i]$.
- **Linearly Constrained Program:** The constraint functions g and h are linear, but the objective function f may be nonlinear. There are no integer or binary constraints. The set B must be of the form $B = \prod_{i=1}^n [a_i, b_i]$. Linear constraints are easier to work with than nonlinear constraints, since the feasible set is a polyhedron, which is convex, and the optimality conditions are easier fulfilled on convex sets. For example, if the feasible set is convex and the objective function is convex, then all local minima are global minima.
- **Quadratic Program:** The constraint functions F and H are linear, but the objective function f is quadratic. Quadratic programs also require that the set B take the form $B = \prod_{i=1}^n [a_i, b_i]$.
- **Nonlinear Optimization Problem:** A nonlinear optimization problem (NLP) does not require the objective function or the constraints to be linear. It has no integer or binary constraints.
- **Integer Optimization Problem:** An optimization problem where all variables are required to be integers: $x \in \mathbb{Z}$.
- **Mixed Integer Optimization Problem:** The functions f , F , and H are all linear, but at least some of the variables are required to be integers. If necessary, integer constraints can be transformed to regular constraints: i.e. $x \in \mathbb{Z}$ is equivalent to $\sin(\pi x) = 0$.

- **Mixed Integer Nonlinear Problem:** The functions f , F , and H can be non-linear, and there can be integer constraints.

It is interesting to note that the words “problem” and “program” are both used to describe optimization problems. The word “program” in this case means succession of events, like at a well-planned party or a conference. Sadly enough, it comes from military programming, which has to do with getting combat units ready for battle. Originally, when he came up with the term in 1947, George Dantzig had called linear programming “Programming in a Linear Structure,” since he was working on an Air Force planning problem, but then he changed the name to “Linear Programming” while on the beach with a friend (which is nice in comparison to war.) [Dantzig]

2.2 Local Optimization

There are many ways to formulate optimization problems, and most of them are equivalent mathematically. However, certain problem formulations allow certain algorithms to run more efficiently, so problems should be formulated in ways which allow the algorithm being used to be most efficient. For example, an equality constraint can easily be converted into a pair of inequality constraints:

$$F(x) = 0 \Leftrightarrow F(x) \leq 0 \wedge F(x) \geq 0.$$

Converting from inequality constraints to equality constraints is more difficult and involves the addition of **slack variables** to the problem:

$$F(x) \leq 0 \Leftrightarrow F(x) + s = 0 \text{ for some } s \geq 0.$$

Notice that with the above transformation, we eliminated an equality constraint, but gained a bound constraint.

A note to notation:

From now on, we will always refer to the gradient $\nabla f(x)$ of the objective function as $g(x)$ and the Hessian $\nabla^2 f(x)$ of the objective function as $G(x)$. The general form of optimization problems being considered in a section will be introduced in that section, but it is important to remember that most optimization problems can be written in many different formats.

2.2.1 Lagrangian, Optimality Conditions, and Duality

2.2.1.1 The Lagrangian

Unconstrained optimality problems can generally be solved by the strategy we learned at school: Calculate the derivative (or the gradient in several dimensions), set it to zero, and then use the Hessian to determine if the roots are minima, maxima, or saddle points. Constrained problems, however, require more effort, since the extrema found with the above method might not lie in the feasible set, and the extrema of the constrained problem may only be extrema because they lie on the boundary of the feasible set, and not because their gradient is equal to zero.

The Lagrange method allows us to convert equality constrained problems into unconstrained problems, and then find the minima (or maxima) with the classic method for unconstrained problems. Consider a problem of the form:

$$\begin{aligned} \min f(x) \\ \text{s.t. } F(x) = 0 \\ x_i \in [a_i, b_i], \end{aligned} \tag{2.1}$$

with objective function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ and constraint functions $F : \mathbb{R}^n \rightarrow \mathbb{R}$. In this case, the minima must lie on the surface of $F(x) = 0$, and the value of f is extremal when the surface of the objective function touches the surface of the constraint functions, that is, when the two surfaces share a tangent line. Therefore, the gradient of the objective and the constraints are parallel at an extremum, that is: $\nabla f = \lambda \nabla g$. This actually only holds if F' has full rank, and, because of this restriction, the Lagrange Multiplier Rule includes the constraint qualification $\text{rank}(F'(x)) = m$, for $F : U \rightarrow \mathbb{R}^m$.

The Lagrangian method is used to transform a constrained problem of the form (2.1) with n variables and m constraints into an unconstrained problem. It does this by adding multiples of the equality constraints F_k , $k \in \{1, \dots, m\}$ to the objective function to form the **Lagrangian**:

$$L(x, y) = f + \sum_{k=1}^m y_k F_k$$

The new function has $(n + m)$ variables. The new variables, y_k , are known as **Lagrange multipliers**. Finding the critical points of the Lagrangian is an unconstrained

problem, making it (generally) easier to solve than the original problem.

Theorem: Lagrange Multiplier Rule:

Let $U \subseteq \mathbb{R}^n$ be open, $f : U \rightarrow \mathbb{R}$, $F : U \rightarrow \mathbb{R}^m$, f, F , be $\mathcal{C}^1$. Let $\hat{x}$ be a local minimum of the optimization problem (2.1).

Then there are $\kappa \geq 0$, $\hat{y} \in \mathbb{R}^m$ (not both = 0), such that

$$\kappa g(\hat{x}) + F'(\hat{x})^T \hat{y} = 0. \quad (2.2)$$

If $F'(\hat{x})$ has full rank, then $\kappa = 1$, that is: $g(\hat{x}) + F'(\hat{x})^T \hat{y} = 0$.

Equation (2.2) is equivalent to $g(\hat{x}) = -\frac{\hat{y}}{\kappa} \cdot F'(\hat{x})$, which means that the gradient of the objective is a linear combination of the gradients of the constraint function. In several dimensions, (2.2) means that $g(\hat{x}) \in \text{span}\{F'_1(\hat{x}), \dots, F'_m(\hat{x})\}$.

If we differentiate the Lagrangian, we get:

$$\nabla L(x, y) = \begin{pmatrix} \nabla_x L \\ \nabla_y L \end{pmatrix} = \begin{pmatrix} g(x) + F'(x)^T y \\ F(x) \end{pmatrix}.$$

A critical point $(\hat{x}, \hat{y})$ of $L(x, y)$ satisfies $\nabla L(x, y) = 0$. Therefore:

$$\begin{aligned} g(\hat{x}) + F'(\hat{x})^T \hat{y} &= 0 \\ F(\hat{x}) &= 0 \end{aligned} \quad (2.3)$$

That means $(\hat{x}, \hat{y})$ satisfies the equality constraints of (2.1), so it is a feasible point. We know that (2.3) is true for all local minima of f , at least if F has full rank. Thus, we have a system of $(m + n)$ equations which we can solve to find $\hat{x}$ and $\hat{y}$.

Consider the following example, which is a simplification of the milk maid problem. [Jensen, 2008]

Florian is on a camping trip with his brother Philipp. He has been scavenging for nuts and berries, when Philipp calls him to dinner. Floh's hands are dirty from all the scavenging, so he wants to go to the river and wash his hands quickly, before joining Philipp at the campfire. He knows Philipp is impatient to eat, and Floh is very hungry himself, so he really wants to take the quickest path possible. He decides to work out an optimization problem to find the shortest path between the tree where he was looking for nuts, the river, and the campfire, since he is convinced that the time he saves by following the shortest path (even though there are no trees or anything in

his way) will more than make up for the time he spends calculating. Floh is known for being able to calculate things quickly in his head, so he really might not be wasting time. Anyway, he draws a diagram with a stick in the dirt.

He then formulates his optimization problem as:

$$\begin{aligned} \min_R \quad & \| \text{tree} - R \|^2 + \| \text{campfire} - R \|^2 =: F(R) \\ \text{s.t.} \quad & \text{river}(R) = 0 \quad (\Leftrightarrow R \text{ is on the river}). \end{aligned}$$

Floh knows that the distance from one focal of an ellipse to any point on the ellipse and then to the other focal is always constant, so he decides to consider the tree and the campfire as foci, and trace ellipses of various sizes around the tree and the fire. This is very smart of Floh, since the ellipses describe constant values of F ; they are niveau lines of F . Floh finds the smallest ellipse that touches the river, and takes off in the direction of the intersection. Interestingly, the smallest ellipse and the river touch at exactly one point: they are tangential. Thus, the direction of the river at the optimal point $\hat{R}$ is parallel to the direction of the ellipse (the niveau line of F) at that point. That is, the gradient of the river is a multiple of the gradient of the niveau line of F at the optimal point; in the terminology from before, we have $g(\hat{x}) = a \cdot F'(\hat{x})$, just like in the Lagrange Multiplier Rule, where a is a constant.

The Lagrange Multipliers are basically just artificial variables added to the problem to make it easily solvable, and do not need to be interpreted further than that. However, in some situations they have interesting interpretations. For example, in variational calculus, a constraint's multiplier represents the force needed to impose that constraint. This can also be seen from the definition of the Lagrangian:

$$L(x, y) = f + \sum_{k=1}^m y_k F_k$$

The y_k seem to represent the amount of energy each constraint F_k needs to exert in order to pull the objective into its feasibility domain. [Jensen, 2008]

2.2.2 Optimality Conditions

Consider the optimization problem:

$$\begin{aligned}
 & \min f(x) \\
 & s.t. \ C(x) \geq 0 \\
 & \quad F(x) = 0 \\
 & \quad f : \mathbb{R}^n \rightarrow \mathbb{R}, \\
 & \quad F : \mathbb{R}^n \rightarrow \mathbb{R}^r \\
 & \quad C : \mathbb{R}^n \rightarrow \mathbb{R}^m \\
 & \quad f, F, \text{ and } C \text{ are } \mathcal{C}^1
 \end{aligned} \tag{2.4}$$

Theorem: First Order General Optimality Conditions

Let $\hat{x}$ be a solution of the above optimization problem and let C be convex. Then there are $\kappa \geq 0, \hat{y} \in \mathbb{R}^n, \hat{z} \in \mathbb{R}^r$ with:

$$\left. \begin{aligned}
 \kappa g(\hat{x}) &= C'(\hat{x})^T \hat{y} + F'(\hat{x})^T \hat{z} \\
 \inf(\hat{y}, C(\hat{x})) &= 0 \\
 F(\hat{x}) &= 0 \\
 \kappa &= 1 \text{ or } \kappa = 0 \text{ and } \hat{z} \neq 0.
 \end{aligned} \right\} \text{Karush-John Conditions}$$

The Karush-John and the Karush-Kuhn Tucker conditions generalize the Lagrange conditions to more general optimization problems with inequality constraints. They are necessary conditions, meaning that they are true for the point $\hat{x}$, if it is a local minimum. There are also sufficient conditions, which are more powerful because if they are true for a point $\hat{x}$, then $\hat{x}$ is a local minimum. On the other hand, necessary conditions are useful for checking if a point is a minimum quickly, so both conditions have their own strengths. The most powerful conditions are the conditions which are necessary and sufficient.

Theorem: First Order Necessary Optimality Conditions with Constraint Qualification:

Let $\hat{x}$ be a solution of (2.4). If some constraint qualification is fulfilled, for example

Slater's Condition, then there exist $\hat{y} \in \mathbb{R}^m$ and $\hat{z} \in \mathbb{R}^r$ so that

$$\left. \begin{aligned} g(\hat{x}) &= C'(\hat{x})^T \hat{y} + F'(\hat{x})^T \hat{z} \\ \inf(\hat{y}, C\hat{x}) &= 0 \\ F(\hat{x}) &= 0 \end{aligned} \right\} \text{Karush-Kuhn-Tucker Conditions}$$

holds.

There are many possible constraint qualifications, and only one of them has to be fulfilled in order for a point to be a Kuhn-Tucker point. One example is Slater's Condition which requires that the gradients of the active constraints are linearly independent at $\hat{x}$. The active constraints are the inequality constraints which $\hat{x}$ fulfills exactly, that is all constraints $C_i(x)$ with $C_i(\hat{x}) = 0$. Naturally, the equality constraints are considered active in that respect, as well. Another example is the Mangasarian-Fromovitz Constraint Qualification, which is similar to Slater's Condition, but requires that the gradients of the active constraints are positively linear independent.

The second condition, called the complementarity conditions, ensures that we are either on an active constraint ($C(\hat{x}) = 0$) at $\hat{x}$ or the multiplier is zero. The first constraint implies that the gradient of the objective, $g(\hat{x})$ is in the cone spanned by the active constraints. This makes sense geometrically, because the gradients of f and each active F_k must be pointing away from the feasible set, because the function should be decreasing as it leaves the feasible set, otherwise the point found would not be a local minimum, after all.

2.2.2.1 Duality

Consider the problem:

$$\begin{aligned} &\min_x f(x) \\ &s.t. \ F(x) \geq 0 \\ &\quad x \in C \end{aligned}$$

The above problem is called the **primal** problem.

Let f, F be convex $\mathcal{C}^1$ functions and C be a convex set. It holds that:

If there exist $\hat{x} \in C$, $0 \leq \hat{y} \in \mathbb{R}^n$, such that: $g(\hat{x}) + F'(\hat{x})^T \hat{y} = 0$, then

$\min\{f(x) \mid x \in C, F(x) \geq 0\} \geq L(\hat{x}, \hat{y})$. This gives us a lower bound for the minimum of f .

We now formulate the **dual problem**:

$$\begin{aligned} & \max L(x, y) \\ & s.t. \ g(x) + F'(x)^T y = 0 \\ & \quad x \in C, y \geq 0 \end{aligned}$$

The previous statement is equivalent to: if $\hat{x}$ is a solution of the primal problem and $(\bar{x}, \bar{y})$ is a solution of the dual problem, then $L(\bar{x}, \bar{y}) \leq f(\hat{x})$.

The **duality gap** $\Delta := f(\hat{x}) - L(\bar{x}, \bar{y})$ describes the difference between the primal and the dual solutions.

If $\Delta = 0$, then $\bar{x}$ is a solution of the primal problem, too, and because $(\bar{x}, \bar{y})$ is a solution of the dual problem, it holds that $g(\bar{x}) + F'(\bar{x})^T \bar{y} = 0$.

$$\begin{aligned} \Delta = 0 & \Leftrightarrow f(\bar{x}) = L(\bar{x}, \bar{y}) = f(\bar{x}) + \bar{y}^T F(\bar{x}) \\ & \Leftrightarrow 0 = \bar{y}^T F(\bar{x}) \\ & \Leftrightarrow \inf(\bar{y}, -F(\bar{x})) = 0 \end{aligned}$$

The following theorem sums things up:

Theorem: Sufficient Optimality Conditions for Convex Problems:

Let $C \subseteq \mathbb{R}^n$ be convex, $f : C \rightarrow \mathbb{R}$, $F : C \rightarrow \mathbb{R}^n$, f, F convex and continuously differentiable.

If there are $\bar{x} \in C$ and $\bar{y} \in \mathbb{R}^n$ with $g(\bar{x}) + F'(\bar{x})^T \bar{y} = 0$ and $\inf(\bar{y}, -F(\bar{x})) = 0$, then $\bar{x}$ is a global solution of the primal problem and $(\bar{x}, \bar{y})$ is a global solution of the dual problem. The duality gap $\Delta = 0$.

Solving the set of $(n + k)$ equations

$$\begin{aligned} g(\bar{x}) + F'(\bar{x})^T \bar{y} &= 0 \\ \inf(\bar{y}, -F(\bar{x})) &= 0 \end{aligned}$$

for x and y is one way to find a lower bound on the solution of the primal problem,

and in many cases gives a solution.

2.2.3 Descent Methods

Descent methods are a general type of algorithm for finding a local minimum. The goal is to find a sequence of vectors $(x_i)_i$ such that the value of the objective function is descending in each step, i.e. $f(x_{i+1}) < f(x_i)$. Good descent sequences converge to stationary points.

The gradient descent algorithm is an example of a simple descent method. The descent direction at each point x_i is the negative gradient $\nabla f(x_i)$, since that is the direction in which $f(x_i)$ decreases the fastest. The starting value is chosen as x_0 , and in each iteration i , we set $x_i = x_{i-1} + \alpha_i \nabla f(x_{i-1})$. The variable α_i is chosen so that $f(x_i) \leq f(x_{i-1})$.

The theory behind descent sequences becomes more complicated when constraints are taken into account, because the descent sequence has to end up (converge) at a feasible point.

2.2.3.1 Line Search

A **line search** is a method for constructing a descent sequence. In each step, x_{i+1} is defined as $x_i + s(\alpha)$, where α is the **step size**. The variable α is chosen so that $f(x_{i+1}) = f(x_i + s(\alpha)) < f(x_i)$ in each step i . The following **descent condition** must hold for line searches: There exists an $\varepsilon > 0$ such that $g^T s(\alpha) \leq -\varepsilon \alpha$ for all $\alpha \geq 0$. This ensures that the angle between the search direction and the gradient is greater than 90° .

The gradient descent algorithm described above is an example of a line search: For each iteration i , $x_i = x_{i-1} + \alpha_i \nabla f(x_{i-1})$. In this case, $s(\alpha) = \alpha_i \nabla f(x_{i-1})$. To be more precise, this is a linear line search, with **search direction** $p = \nabla f(x_{i-1})$. In general, linear line searches have the form $x_{i+1} = x_i + s(\alpha)$, with $s(\alpha) = p \cdot \alpha$.

In each step of a line search, only the parameter α needs to be determined. This can either be determined exactly, so that it is a global minimum of $f(x + s(\alpha))$, or loosely,

so that $f(x + s(\alpha))$ is significantly less than $f(x)$, but not necessarily the very best choice. Usually α is only chosen loosely, since an exact line search requires more effort.

Bent line searches are used in the case of bound constrained problems. This method projects the path $x + \alpha p$ into the feasible direction, so that the line search does not leave the feasible set and the descent sequence eventually converges to a feasible point.

2.2.3.2 Active Set Methods

Active set methods are used for problems of the type

$$\begin{aligned} \min f(x) \\ \text{s.t. } x \in [u, v]. \end{aligned}$$

In each step, they figure out which constraints are currently active, that is, for which either $u_i = x_i$ or $x_i = v_i$ holds. These x_i are then held constant, so that the optimization takes place on this “active set” $A = \{i \mid u_i = x_i \text{ or } x_i = v_i\}$, and uses descent methods to find the minima in the directions x_j with $j \in I$, where I is the “inactive set:” $I = \{i \mid u_i < x_i \text{ and } x_i < v_i\}$. Thus, the problem is reduced to the problem:

$$\begin{aligned} \min f(x_I, x_A) \\ \text{s.t. } x_I \in [u_I, v_I]. \end{aligned}$$

This is called the **reduced problem**.

If a minimum of the reduced problem is found, then the active set has to be adjusted so that optimization can be continued. Typically, each index is held fixed for a certain number of iterations, so that the algorithm does not keep returning to a face of the problem where there is nothing left to optimize; so that the optimization does not “zig-zag” unproductively.

2.2.3.3 Trust Regions and Newton-like Methods

Taylor's Theorem says that (under certain conditions)

$$f(x + s) = f(x) + g(x)^T s + \frac{1}{2} s^T G(x) s + \mathcal{O}(\|s\|^2).$$

We can use Taylor's Theorem to approximate $f(x)$ quadratically:

$$f(x + s) \approx q(x + s) = f(x) + g(x)^T s + \frac{1}{2} s^T G(x) s.$$

If it is too much work (or not possible) to calculate the Hessian matrix (just calculating it once is not that big of a deal, but optimization algorithms end up calculating the Hessian hundreds or thousands of times), we can approximate the Hessian as $B \approx G(x)$. Consider the following theorem:

Theorem: If $g \neq 0$, then a search direction p satisfies the descent condition $g^T p < 0$ (for the linear line search $s(\alpha) = \alpha p$, $x_{i+1} = x_i + s(\alpha)$), if and only if there is a symmetric, positive-definite matrix B such that $p = -B^{-1}g$.

For a quasi-Newton Method, we will assume that B is symmetric and positive definite, and choose $p = -B^{-1}g$ as the search direction. We then look for the minimum of $q(x + s)$ instead of the minimum of $f(x + s)$. Newton's Methods use the Hessian matrix itself to perform the line search: $p = -G(x)^{-1}g(x)$.

We still need a step direction, and have to be careful that we do not step so far that the quadratic approximation loses its validity. We let σ be the trust region radius, the region in which we can trust the quadratic approximation. Then we can choose the step length by either restricting steps to the area $|\alpha p| \leq \alpha$, or by solving the quadratic problem:

$$\begin{aligned} \min q(x + s) \\ \text{s.t. } \|s\| \leq \sigma \end{aligned}$$

This problem is actually convex, if B is positive definite, and can be solved in $\mathcal{O}(n^3)$ if the 1-norm or ∞ -norm is used.

2.2.4 Penalty methods and Interior Point/Barrier Methods

These methods transform constrained problems into unconstrained problems. Consider a constrained problem of the form:

$$\begin{aligned} \min f(x) \\ \text{s.t. } F(x) = 0, a \leq x \leq b. \end{aligned}$$

Penalty methods typically minimize the penalty function $f_\sigma(x) := f(x) + \sigma\phi(\|F(x)\|)$, or a similarly-styled function, which adds constraint violations into the objective function. In each iteration, the bound constrained penalty problem:

$$\begin{aligned} \min f_\sigma(x) \\ \text{s.t. } a \leq x \leq b \end{aligned}$$

is solved, and then the penalty parameter $\sigma > 0$ is increased. Eventually, the sequence of solutions of the penalty problems will converge to the solution of the original problem, because for large σ , the value of f_σ is only minimal for small $\sigma\phi(\|F(x)\|)$, so f_σ is minimal when the original constraints are not violated.

Interior point methods also add penalties in each iteration, but the value of the penalty function rises to infinity as the border of the feasible set is approached. This ensures that feasible points are found in each iteration. In this case, the term “barrier function” is sometimes used instead of “penalty function.”

Interior point methods can be slower than penalty methods, but, since they find relatively feasible points in each iteration, they can be stopped earlier than penalty methods, if a person needs a reasonably good solution quickly. Unfortunately, they can not be used on equality constrained problems, since these problems do not have any interior points.

2.3 Discrete Optimization

If one of the constraints of an optimization problem is that the solution must be in $\mathbb{Z}^n$, then the optimization process must be carried out differently. Since the domain of the objective function is then a subset of $\mathbb{Z}^n$, the function is not differentiable.

In addition, since $x \in \mathbb{Z}^n$, it is possible to find a neighborhood around every point x , so that it is a minimum of $f(x)$ in that neighborhood. Therefore, to find a less trivial minimum, a global solution must be sought. Various methods are used to solve discrete optimization problems. I will only discuss dynamical programming methods here.

2.3.1 Hidden Markov Models and Dynamical Programming

This section is based on Rabiner [1989], Moore, and Jones and Pevzner [2004].

Definition: A **Markov System** is a system with N states $S_1, \dots, S_N$. At each time step t in a series of discrete time steps, the system is in exactly one state, $q_t \in S_1, \dots, S_N$. The set of all possible state sequences is $\mathcal{Q} = \{Q = \{q_1, \dots, q_N\} \mid q_i \in S_1, \dots, S_N\}$. The **Markov Property** is that the state q_{t+1} is conditionally independent of the states $q_1, \dots, q_{t-1}$, that is:

$$P(q_{t+1} = S_i \mid q_t = S_k, q_{t-1} = S_l, \dots, q_1 = S_r) = P(q_{t+1} = S_i \mid q_t = S_k).$$

In other words, q_{t+1} depends only on the immediately preceding state, q_t . The state transition probability matrix $A = (a_{ij})$ is given as: $a_{ij} = P(q_t = S_j \mid q_{t-1} = S_i)$, for $1 \leq i, j \leq N$.

Definition: In a **Hidden Markov Model (HMM)**, the observable states depend on a set of states which are not observable, but hidden (hence the name). The sequence of hidden states, $Q = q_1, \dots, q_N$ is a stochastic process with the Markov Property. The state transition probabilities from the state $q_t = S_i$ to the state $q_{t+1} = S_j$ are given by the matrix A described above, and the probability of the observation v_k from the state S_i is given by $b_i(k)$, that is, $b_i(k)$ is the conditional probability $b_i(k) := P(k \mid q_k = S_j)$. For the M possible observations $v_1, \dots, v_M$, the matrix $B = (b_j(k))$, for $1 \leq j \leq N$, and $1 \leq k \leq M$ describes the observation probability distribution. The probabilities for the value of the initial state are given by $\pi_i = P(q_i = S_1)$, $i = 1, \dots, N$. A Hidden Markov Model is completely described by $\lambda = \{N, M, \{\pi_i\}, A, B\}$. The number of timesteps, T , which is equal to the number of observations $O_t \in \{v_1, \dots, v_M\}$, is important for working with Hidden Markov Models, but is independent of the model.

Example: Consider the following example, which is a simplification of the “Fair Bet Casino Problem” described in Jones and Pevzner [2004]: Teddy is sitting in his room

tossing one or more coins. I don't know how many coins, and I don't know if they are biased or not. He keeps exclaiming "Heads" and "Tails" seemingly randomly. I have studied math for almost five years now, so I think I can figure out what he's doing without just asking him. So far, he's said: "H H H T H H H T T H H T T H T H T T H T T T T T T H T T T." (H = heads, T = tails.) So I decide to make a Hidden Markov Model. I declare the hidden states to be the different coins being used. My first guess is that he's using exactly one fair coin. So:

$N = 1$, the number of hidden states.

$M = 2$, the number of possible observations (either heads or tails).

$A = a_{11} = 1$, (always stay in the same state, since there is only one state)

$B = (.5, .5)$, the probability of each observation from the one state.

$\pi_1 = 1$, since I don't have any other options.

With the parameters listed above, the Hidden Markov Model is $\lambda = \{N, M, 1, 1, (.5, .5)\}$. This isn't a very exciting Hidden Markov Model, though, since nothing is actually hidden.

So, I try to make a more interesting, and (hopefully) more accurate, model. This time, I assume that Teddy has three coins: one fair coin, one coin that is biased towards heads, and one coin that is biased to tails. I set initial probabilities π_i and come up with probabilities a_{ij} for how likely he is to switch coins. The probability $b_i(k)$ of the observation v_k from the state S_i is determined by the bias of the coins. This yields a new Hidden Markov Model $\lambda = \{N, M, \{\pi_i\}, A, B\}$, where the number of hidden states, N , is now 3 and the number of possible observations, M , is still 2. This leads us to the three main questions that have to do with Hidden Markov Models:

1.) The first question I would ask is: How likely is Teddy's sequence? Or, more formally posed: Given the observation sequence $O = O_1, \dots, O_T$, what is $P(O|\lambda)$ and how can this probability be computed efficiently?

2.) Another question I might ask is: Which coin was Teddy using at which time? From looking at the observation sequence

"H H H T H H H T T H H T T H T H T T H T T T T T T H T T T",

I think he started out with the head-biased coin, then switched to the fair coin, and

then used the tails-biased coin, but I am not sure when he switched. Thus, the second problem is: Given a sequence of observations $O = O_1, \dots, O_T$, what is the most likely sequence of hidden states, q_i ? That is, what is $\arg \max_Q P(Q|O)$, and how can it be calculated efficiently?

3.) The last thing I would want to do is to adjust the Hidden Markov Model so that it best describes the observations. I originally chose the three hidden states to be a heads-biased coin, a tails-biased coin, and a fair coin, because it looks like he might have used those coins. But I have to guess at the actual bias of the coins, and the likelihood of Teddy switching coins. So, the question is: Which HMM models this situation the best; how can $\lambda^* = \arg \max_\lambda P(O|\lambda)$ be computed?

The first two questions can be answered slowly and surely by trying out every possible sequence and comparing the probabilities. However, the computational effort required to do so is enormous. Dynamical programming algorithms are filled with tricks which allow the questions to be answered just as accurately with much less time and effort.

2.3.1.1 Probability of a Series of Observations $O = O_1, \dots, O_T$

We want to know $P(O) = P(O_1, O_2, \dots, O_n)$. The slow way of solving this problem would be to look at every possible state sequence $Q \in \mathcal{Q}$ and find the probability of the series of observations O given that sequence. Then we could calculate $P(O|\lambda)$ as $P(O|\lambda) = \sum_{Q \in \mathcal{Q}} P(O|Q, \lambda) \cdot P(Q|\lambda)$. Calculating $P(O|\lambda)$ this way would require $\mathcal{O}(TN^T)$ operations, which would be horribly inefficient. The trick to finding $P(O)$ efficiently is in thinking that if the partial sequence $O_1, O_2, \dots, O_{t+1}$ is optimal, then the partial sequence $O_1, O_2, \dots, O_t$ is optimal, as well. The following procedure, known as the *Forward Algorithm* finds $P(O|\lambda)$ with only $\mathcal{O}(N^2T)$ operations:

Define $\alpha_t(i) = P(O_1, O_2, \dots, O_t \wedge q_t = S_i \mid \lambda)$, for $1 \leq t \leq T$. The value $\alpha_t(i)$ gives the probability that the first t observations were $O_1, O_2, \dots, O_t$ and the current state is q_t . Since a_{ij} is the probability that $q_{t+1} = S_j$, given $q_t = S_i$, then $\alpha_t(i) \cdot a_{ij}$ is the probability that $O_1, O_2, \dots, O_t$ was observed, $q_t = S_i$, and $q_{t+1} = S_j$. Therefore, we can define

$$\alpha_{t+1} = P(O_1, O_2, \dots, O_{t+1} \wedge q_{t+1} = S_j \mid \lambda)$$

and it holds that¹

$$\alpha_{t+1} = \sum_{1 \leq i \leq N} (\alpha_t(i) \cdot a_{ij}) \cdot b_j(O_{t+1}).$$

This leads to the following iterative procedure (Rabiner [1989]):

- 1.) Initialize $\alpha_1(i) = \pi_i \cdot b_i(O_1)$ for $1 \leq i \leq N$.
- 2.) For $1 \leq j \leq N$: $\alpha_{t+1} = \sum_{1 \leq i \leq N} (\alpha_t(i) \cdot a_{ij}) \cdot b_j(O_{t+1})$ This requires N calculations for each j . Since there are N states S_j in each time step t , this makes for $N \cdot N$ calculations. This is done for $t = 1, \dots, T-1$, so for a total of $T \cdot N^2$ calculations.

For $t = T$, it holds that: $\alpha_T(i) = P(O_1, \dots, O_T \wedge q_T = S_i \mid \lambda)$. Therefore,

$$P(O \mid \lambda) = \sum_{i=1}^n P(O_1, \dots, O_T \wedge q_T = S_i \mid \lambda) = \sum_{i=1}^n \alpha_T(i).$$

This same probability can be computed analogously backwards by using the backwards variables: $\beta_t(i) = P(O_{t+1}, O_{t+2}, \dots, O_T \wedge q_t = S_i \mid \lambda)$, $1 \leq t \leq T$, and by initializing $\beta_T(i)$ instead of $\alpha_1(i)$.

2.3.1.2 Most likely hidden state sequence, given a series of observations

Typically, HMM's are implemented in order to find the most likely hidden state sequence, given a sequence of observations. For example, HMM's are commonly used to allow machines to figure out what a person is saying by hanging together the word parts that the machine is able to record. In this case, the actual words are the hidden states, and the word parts are the observations.

The first difficulty in solving this problem is defining the criteria for the “most likely” sequence. We will consider the best sequence to be the sequence of states $Q = q_1, \dots, q_T$ with the highest joint probability, since this definition most closely matches the intentions of this paper. The *Viterbi Algorithm* solves the problem of finding the sequence Q such that

$$Q = \arg \max P(Q \mid O, \lambda).$$

It works by first finding the probability of the most probable sequence $Q^* = q_1, \dots, q_T$,

¹see Moore

and then finding the sequence itself through backtracking. It begins by defining $\delta_t(i)$, the probability of the most likely partial state sequence $mpp_t(i) = q_1, \dots, q_{t-1}$, with $q_t = S_i$, so that the first t observations are $O_1, \dots, O_t$:

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} P(q_1, q_2, \dots, q_{t-1} \wedge q_t = S_i \wedge O_1, \dots, O_t \mid \lambda)$$

Then the algorithm takes a step forward to $(t+1)$. To do this, the most probable path to $q_{t+1} = S_j$ must be found. The most probable path to $q_t = S_i$ for all $i = 1, \dots, N$ is known, so the most probable path to $q_{t+1} = S_j$ can be calculated as the path $\{mpp_t(i), S_j\}$, such that $\delta_t(i) \cdot a_{ij}$ is maximal over all i . Thus, $\delta_{t+1}(j)$ is the probability of the most likely path to $q_{t+1} = S_j$, such that the observation sequence is

$$O_1, \dots, O_t, O_{t+1} = (\max_{1 \leq i \leq N} \delta_t(i) \cdot a_{ij}) \cdot b_j(O_{t+1}).$$

A backtracking step is performed to find the optimal state sequence directly after the Viterbi Algorithm, since the algorithm only finds the probability of the optimal sequence, but conveniently stores the information needed to find the optimal state sequence itself. The algorithm below can be found in Rabiner [1989]:

The Viterbi Algorithm:

1.) Initialize $\delta_1(i)$ for $i = \{1, \dots, N\}$ based on the initial probabilities of each state and the first observation:

$$\begin{aligned} \delta_1(i) &= \pi_i \cdot b_i(O_1) \\ \psi_1(i) &= 0. \end{aligned}$$

2.) For $2 \leq t \leq T$ and $1 \leq j \leq N$, do the following:

$$\begin{aligned} \delta_t(j) &= (\max_{1 \leq i \leq N} \delta_{t-1}(i) \cdot a_{ij}) \cdot b_j(O_t) \\ \psi_t(j) &= \arg \max_{1 \leq i \leq N} (\delta_{t-1}(i) \cdot a_{ij}). \end{aligned}$$

3.) Find the maximal probability P^* :

$$\begin{aligned} P^* &= \max_{1 \leq i \leq N} (\delta_T(i)), \\ \text{and find the optimal value of hidden state } q_T: \\ q_T^* &= \arg \max_{1 \leq i \leq N} (\delta_T(i)) \end{aligned}$$

4.) Find the rest of the optimal state sequence through backtracking:

Starting with $t = T - 1$ and working backwards towards $t = 1$:

$$q_t^* = \psi_{t+1}(q_{t+1}^*).$$

2.3.1.3 Training the Hidden Markov Model

Training the Hidden Markov Model, that is, improving the parameters, has more to do with data analysis and is described nicely in Jones and Pevzner [2004], and will be explained specifically for this project in Hattinger [2008].

2.3.1.4 Connection to Protein Folding

In Section 3.3, our use of HMM's for protein folding is explained in detail, but the basic connection is explained here. We want to predict the tertiary structure of a protein with n amino acids. To do this, we develop a set of n_g classes which describe the local structure of the protein based on the geometry and composition of each fragment of four consecutive amino acids ("4-fragment.") We also develop a set of γ -classes which can be derived directly from the amino acid sequence, so its members are visible to us. These classes describing the local structure of each of the $m = (n - 3)$ four-fragments. These classes are developed with knowledge of the secondary structure classes, and therefore are intrinsically related to the secondary structure classes². Our goal in using Hidden Markov Models is to perform discrete optimization in order to find a set of the most likely secondary structure class sequences $s_1, \dots, s_m$, given the primary structure class sequence $\gamma_1, \dots, \gamma_m$. The sc -classes are the model's hidden states, and the γ -classes are the model's observations. The probabilities $b_j(k) = P(\gamma = k | s_t = j)$ are assumed to be $-\log(\text{confus}(k, j))$, where confus is a $(n_g \times n_{sc})$ matrix with

$\text{confus}(i, j) = \{ \text{the number of occurrences of a 4-fragment of class } \gamma = i \text{ and } sc = j \}.$

The probabilities $a_{ij} = P(s_t = j | s_{t-1} = i)$ are easy to calculate, since it is either possible to go from one sc to another or it is not, so $a_{ij} = \begin{cases} 1 & \text{if } s_i \rightarrow s_j \\ 0 & \text{otherwise.} \end{cases}$

Using these parameters, our Hidden Markov Model is:

$$\lambda = \{n_{sc}, n_g, -\log(\text{confus}(1, j)), A, B\}$$

²The classification of these classes uses the sc -classes as its initial class set. See Hattinger [2008].

2.4 Global Optimization

While local optimization is concerned with finding any feasible solution of an optimization problem, global optimization tries to find the overall minimum or maximum. This is usually difficult, so some kinds of stochastic methods are applied in order to find the solution of the problem using a reasonable amount of time and effort. Most global optimization problems need exponential time (and effort) to solve [Schichl, 2001]. Global optimization may be necessary if any minimum will not suffice, that is, if the true minimum is needed to solve a problem. One of the concerns of this project is to compare the local optima found by starting the optimization program at various starting points, to see if the function has several feasible minima, if some minima are preferable than others, etc. This information will allow us to decide if global optimization is necessary in order to make accurate protein structure predictions using the methods described in this project.

3 Methods

We are using the following model to predict a protein’s structure from its amino acid sequence: We assume that the protein has its ideal structure at the minimum of our potential function V . Therefore, we find the minimum of V . The parameters of V are determined by the protein’s amino acid sequence. To consider the effects of local interactions between specific amino acids, we created n_γ different classes which describe all possible chains of four amino acids (4-fragments). The geometry of a specific 4-fragment is used to determine its primary sequence class (γ -class). Using a covariance model, parameters for the potential $V_\gamma(a_{i:i+3})$ specific to each γ -class were created (see Hattinger [2008]), where $a_{i:i+3}$ refers to the four-fragment $a_i, a_{i+1}, a_{i+2}, a_{i+3}$, with the i^{th} amino acid in the protein’s primary sequence represented by a_i . Therefore, our potential function is the sum of the potentials of each of the 4-fragments in the protein. The model’s constraints take global aspects of the protein’s structure into account:

$$\begin{aligned} \min \sum_{i=1}^{n-3} V_\gamma(a_{i:i+3}) \\ \text{s.t constraints describing global behavior are fulfilled} \end{aligned}$$

3.1 Modeling the Protein’s Geometry

The values c_i , c_{i+1} , and z_i describe the geometry of each of the $(n - 3)$ 4-fragments. These values are used in place of the cartesian coordinates for the classification and the optimization because they are rotation- and translation-invariant, so they describe the geometry of any 4-fragment at any position in any protein. Before calculating the geometry, we normalize all the bond lengths between two consecutive amino acids. This is not removing any significant information, since the bond lengths are all approximately equal, anyway. Therefore, we can assume that the bond length $\|a_i - a_{i+1}\|$ is always equal to one.

In a four-fragment, four consecutive amino acids a_i , a_{i+1} , a_{i+2} , and a_{i+3} are connected by three consecutive bond vectors. The angle between the first two bond vectors, r_i and r_{i+1} is called α_i , and the angle between the second two bond vectors, r_{i+1} and r_{i+2} is called α_{i+1} . The torsion of the fragment is described by the angle β_i .

Let $c_i = \langle r_i, r_{i+1} \rangle$, for $i = \{1, 2\}$. Since the magnitude of the bond vectors is always approximately 1, c_i represents the cosine of α_i :

$$\cos(\alpha_i) = \frac{\langle r_i, r_{i+1} \rangle}{\|r_i\| \|r_{i+1}\|} = \frac{\langle r_i, r_{i+1} \rangle}{1 \cdot 1} = \langle r_i, r_{i+1} \rangle = c_i$$

Therefore, the sinus of α_i is given by

$$\sin(\alpha_i) = \sqrt{1 - c_i^2} =: s(i),$$

for $i = \{1, 2\}$. The following formulas define the cosine and the sine of the torsion angle β_i , for $i = \{1, 2\}$:

$$\begin{aligned} t_i &:= \cos(\beta_i) = \frac{r_i \cdot r_{i+2} - c_i \cdot c_{i+1}}{s_i \cdot s_{i+1}} \\ t'_i &:= \sin(\beta_i) = \frac{\det(r_i, r_{i+1}, r_{i+2})}{s_i \cdot s_{i+1}} \end{aligned}$$

We used the following system to define z_i and determine the secondary structure class of the i^{th} 4-fragment simultaneously.

The cosine and sine of the torsion angle β_i can be calculated as follows: Let

$$\begin{aligned} \cos(\beta_i) &= r_i \cdot r_{i+2} - c_i \cdot c_{i+1} \\ \sin(\beta_i) &= \det(r_i, r_{i+1}, r_{i+2}) \end{aligned}$$

We let z_i be either equal to $\sin(\beta_i)$ or $\cos(\beta_i)$ depending on the following criteria, which we also use to determine the secondary structure class of the fragment:

$$z_i = \begin{cases} \sin(\beta_i) & \text{if } |\cos(\beta_i)| \geq |\sin(\beta_i)| \\ \cos(\beta_i) & \text{otherwise} \end{cases}$$

$$s_i = \begin{cases} 1 & \text{if } \cos(\beta_i) \geq |\sin(\beta_i)| \\ 2 & \text{if } \sin(\beta_i) \geq |\cos(\beta_i)| \\ 3 & \text{if } -\cos(\beta_i) \geq |\sin(\beta_i)| \\ 4 & \text{if } -\sin(\beta_i) \geq |\cos(\beta_i)| \end{cases}$$

Therefore, we have four classes $s_i \in \{1, 2, 3, 4\}$, which are also called H, E, G , and F . The first class, H , contains mainly helices and the second class, E , contains mainly β -sheets. The two remaining classes contain a mixture of helices, sheets, and loops.

Originally, we had defined z_i to parametrize the torsion angles based on the classes of c_i and c_{i+1} . Secondary structure classes $s_{c_i} \in \{1, 2, 3\}$ and $s_{c_i^+} \in \{1, 2, 3\}$ were assigned to each c_i and c_{i+1} based on their values. We set

$$\phi = \tan(\nu),$$

where ν described the angle between t_i and t'_i . The combination of the s_c and the s_{c^+} class gave definitions for the classes $s_\phi \in \{1, 2, 3\}$ or $\{1, 2\}$ of ϕ_i as regions along a circle. The center of each of these regions was labeled α_i , and then we defined z_i as:

$$z_i = \tan\left(\frac{\phi_i - \alpha_i}{2}\right).$$

The sec class was then determined as a linear combination of the c_i , c_{i+1} , and ϕ_i class:

$$s_i = s_{c_i} + 3 \cdot s_{c_i^+} + 9 \cdot s_{\phi_i} - 12.$$

Because of the dependence of s_i on c_i and c_{i+1} , not every sequence of s_i 's was possible: The c class of s_{i+1} had to be equal to the c^+ class of s_i , for the transition $s_i \rightarrow s_{i+1}$ to be possible. There were a total of n_{sec} such classes. In this case, we multiplied the values of c_i and c_{i+1} by 100 so that they were integers between -100 and 100 . We later switched to a different classification system, which required a different definition of z_i , for reasons discussed in 4.

3.2 Algorithm Summary

To predict the tertiary structure of a protein from its primary sequence, we do the following:

Phase I: Discrete Optimization

- 1.) Use dynamical optimization to find the optimal secondary structure class sequence and the potential of the partial sequence $s_1, \dots, s_k$ with minimal potential, for $k \leq (n - 3)$ and $s_i \in \{1, \dots, n_{sec}\}$, given the primary structure class sequence γ , which is derived directly from the primary structure.
- 2.) Find a reasonably-sized set S of possible sequences using the partial sequence potentials recorded in Step 1.

Phase II: Continuous Optimization

- 3.) Perform the continuous optimization problem using each of the secondary structure class sequences $s \in S$ found in Step 2 and the primary class sequence γ .
- 4.) Select the structures with the lowest potentials scores as the most likely tertiary structures.

To summarize, we start with the protein's primary sequence, $a_1, a_2, \dots, a_n$. This gives us the primary structure class sequence $\gamma_1, \dots, \gamma_{n-3}$. In Phase I, we find a set S of the most likely secondary structure class sequences $s = s_1, \dots, s_{n-3}$, given the primary class sequence. The predicted secondary structure class sequence and the primary structure class sequence are then used as input in the continuous optimization program in Phase II, where we try to find the geometry values c_i and z_i which minimize the potential V . The solution of this problem gives us the geometry values of the protein's predicted structure, from which we can calculate the C_{alpha} coordinates.

3.3 Phase I: Discrete Optimization

The goal of the discrete optimization phase is to find a set of the most likely secondary structure class sequences $s = s_1, s_2, \dots, s_m$, for $s_i \in \{1, \dots, n_s\}$, $i = \{1, \dots, m\}$, and $m = (n - 3)$, for a protein of length n with "primary" class sequence¹ $\gamma = \gamma_1, \gamma_2, \dots, \gamma_n$. Since s_i can only take on integer values, this is a discrete problem, and since we want to find the best sequence, it is a discrete optimization problem. The following potential function is used to determine which sequence is the "best" or the "most likely":

$$V_{seq}(s) = - \sum_{i=1}^m \log \text{confus}(\gamma_i, s_i)$$

¹the primary and secondary sequence classes are defined in 3.1

The confus matrix is a $(n_\gamma \times n_s)$ matrix, where each entry $\text{confus}(j, k)$ contains the number of 4-fragments in both the primary structure class $\gamma = j$ and secondary structure class $sc = k$ in the entire data set. So the discrete optimization problem is:

$$\begin{aligned} \min_s & V_{seq}(s) \\ \text{s.t. } & s_i \in \{1, \dots, n_{sec}\} \\ & s_i \rightarrow s_{i+1} \text{ is possible} \end{aligned}$$

Because our objective functions for the continuous and discrete optimization programs are not perfect, we do not know that the secondary class sequence with the lowest potential will be the sequence which provides the best tertiary structure after the continuous optimization. Thus, we must look for a set S of the best sequences.

The most straightforward way of finding the minimum sequence would be to try out all class sequences of length m . Since there are n_{sec} secondary structure classes, there are $\underbrace{n_{sec} \cdot n_{sec} \cdot \dots \cdot n_{sec}}_m = n_{sec}^m$ possible sequences (or there would be, if it were possible to go from each class to every other class.) Calculating the potential of one sequence requires m calculations of $-\log \text{confus}(\gamma_i, s_i)$, which is actually trivial because the values are stored in a table, and $(m - 1)$ additions, which makes around $2m$ operations. Therefore, to calculate the potential of all n_{sec}^m sequences, in order to find the most likely sequence, one must do approximately $2 \cdot m \cdot n_{sec}^m = \mathcal{O}(m \cdot n_{sec}^m)$ operations. For a protein of length 103 amino acids and 27 available sc -classes, that makes $2 \cdot 100 \cdot 27^{100} = 2.7378 \cdot 10^{145}$ operations, which is too many to be practical. Using dynamical programming, we can find the best sequence with much less effort.

We start by finding the potential value of all minimal partial sequences starting from the left:

$$\min_{s_1, \dots, s_j} \sum_{i=1}^j V(\gamma_i, s_i), \quad j = 1, \dots, m,$$

and all minimal partial sequences starting from the right:

$$\min_{s_k, \dots, s_m} \sum_{i=k}^m V(\gamma_i, s_i), \quad k = 1, \dots, m.$$

The method for finding these values is very similar to the Viterbi algorithm. We record the potential values calculated in each iteration, because they are the potential values

of each partial sequence, and we use these values to find the next best sequences. We first perform a forward calculation, which finds the potential of each partial sequence, starting from the left:

Viterbi	dynoptL
Hidden Markov Model: timesteps $t = 1, \dots, T$ possible states $S_1, \dots, S_N$ state sequence $Q = q_1, \dots, q_T$ observation sequence $O = O_1, \dots, O_T$ $a_{ij} = P(q_{t+1} = S_j q_t = S_i)$ $b_j(k) = P(v_k \text{ at } t q_t = S_j)$	4-fragments $i = 1, \dots, m$ possible classes $s = 1, \dots, s = n_{sec}$ secondary class sequence $s_1, \dots, s_m$ primary class sequence $\gamma_1, \dots, \gamma_m$ $a_{ij} = \begin{cases} 1 & \text{if } s_i \rightarrow s_j \\ 0 & \text{otherwise} \end{cases}$ $b_j(k) = V(\gamma_{t+1}, k) = -\log(\text{confus}(\gamma_{t+1}, k)), s_i = k$
Algorithm: $\delta_t(i) = \max_{q_1, \dots, q_{t-1}} P(q_1, \dots, q_{t-1} \wedge q_t = S_i, O_1, \dots, O_t)$ $\delta_{t+1}(j) = (\max_{1 \leq i \leq N} \delta_t(i) \cdot a_{ij}) \cdot b_j(O_{t+1}), \forall j = 1, \dots, N$	$\delta_t(i) = \min_{s_1, \dots, s_{t-1}} V(s_1, \dots, s_{t-1} \wedge s_t = i, \gamma_1, \dots, \gamma_t)$ $\delta_{t+1}(j) = (\min_{1 \leq i \leq n_{sec}} \delta_t(i) + a_{ij}) \cdot V(\gamma_{t+1}, j), \forall j = 1, \dots, n_{sec}$
Initialization: $\delta_1(j) = \pi_j b_j(O_1)$	$\delta_1(j) = V(\gamma_1, j)$
Recursion: $\delta_t(j) = (\max_{1 \leq i \leq N} \delta_{t-1}(i) \cdot a_{ij}) \cdot b_j(O_t), \forall j = 1, \dots, N, t = 2, \dots, T$	$\delta_t(j) = (\min_{1 \leq i \leq n_{sec}} \delta_{t-1}(i) + a_{ij}) \cdot V(\gamma_t, j), \forall j = 1, \dots, n_{sec}, t = 2, \dots, m$
Termination: $P_{\max} = \max_{1 \leq i \leq N} (\delta_T(i))$	$V_{\min} = \min_{1 \leq j \leq n_{sec}} (\delta_m(j))$

Using the original classification², a simplification of the algorithm could be made because, although s_i can take on any of the n_s hidden states, a_{ij} is determined only by c_{i+1} , which only has three possible values, $c_{i+1} \in \{1, 2, 3\}$. So, instead of looking at

$$\delta_{t+1}(j) = (\min_{1 \leq i \leq n_{sec}} \delta_t(i)) + a_{ij} \cdot V(\gamma_{t+1}, j), \forall j = 1, \dots, n_{sec},$$

we just look at

$$\delta_{t+1}(j) = (\min_{c_{i+1} \in \{1, 2, 3\}} \delta_t(i)) + a_{ij} \cdot V(\gamma_{t+1}, j), \forall j = 1, \dots, n_{sec},$$

with

$$a_{ij} = \begin{cases} 1 & \text{if } c_{i+1} = c_j \\ 0 & \text{otherwise.} \end{cases}$$

²See 3.1.

The matrix L is recorded in this step, with $L(j, k) = \min \sum_{i=1}^k V_{\gamma_i, i}$ such that $c_{k+1} = j$. With the HEFG classes, c_i is just s_i .

While using the original classes, it was necessary to know the potentials of the partial sequences coming from the right side, as well. The reasons for this are explained below. Therefore, we then did the same thing backwards, and recorded the matrix R in this step, with $R(i, j) = \min \sum_{i=k+1}^n V_{\gamma_i, i}$ such that $c_k = j$. Again, with the HEFG classes, c_i is just s_i .

With the HEFG classes³, the algorithm is even easier, because it is possible to go from any class to any other class, so $a_{ij} = 1$ for all i, j , so L is just a vector, not a matrix, and we do not need R .

So now if we choose any position, i.e. the i^{th} position, in the secondary structure sequence, we can find the minimum potential value of a sequence with any value of s_i at that point:

$$V_i := L_j(c_{i+1}, s_i) - \underbrace{\log(\text{confus}(\gamma_i, s_i))}_{:=V_i(s_i)} + R_k(c_i, s_i)$$

using the values of c_i and c_{i+1} corresponding to s_i . With the HEFG classes, we can replace the matrix R with the variable r_k , since it is possible to go from one class to any other class, so the minimal sequence $s_k, \dots, s_n$ is the same, regardless of the value of s_{k-1} .

By finding $\min_s V_i(s_i)$, we know which s -class is in the minimum sequence at the position i . We can take this value, call it s_i^1 , and go on to find the rest of the minimal sequence.

Originally, we found all the sequences s with $V_{seq}(s) \leq \bar{V}$ in one step. Thus, at this point we made a list of all possibilities for s_i so that $V_i \leq \bar{V}$, where $\bar{V} = V_{min} + \varepsilon$, and then went on to find the rest of each minimal sequence. With this system, if we wanted to find additional sequences later (i.e., after ending the program), we would have to repeat all of the calculation. Therefore, we decided that it would be better to find exactly one sequence at a time, in order of most optimal to least optimal, and store the information needed to go on and find the next sequence without repeating calculations.

³See 3.1.

In any case, in the next step we look for all possible values of s at another position in the sequence, so that the potential value either remains minimal (with the new system) or below a certain $\bar{V}$ (with the original system). That is, we look at either the values of s_{i-1} and s_i , or the values of s_i and s_{i+1} . Calling the upper index k and the lower index j , we now look for values of s_j, s_k that minimize the following potential:

$$V_{j,k}(s_{j:k}) := L_j(s_j) + \left(\sum_{i=j}^k V_i(s_i) \right) + R_k(s_k) \quad (3.1)$$

It holds that:

$$\min\{V_{seq}(s) | s_{j:k} \text{ fixed, constraints as in definition of } V_{seq}\} = V_{j,k}(s_{j:k}).$$

So, by gradually expanding the interval $s_{j:k}$ we can find the minimal feasible sequence, if we take the minimum in each step, or all sequences with $V_{seq}(s) \leq \bar{V}$, if we take all partial sequences $s_{j:k}$ with $V_{j,k}(s_{j:k}) \leq \bar{V}$ in each step. And *that* is how we solve the discrete optimization problem in a feasible amount of time.

Although the improved algorithm, which finds exactly one sequence at a time, uses the basic method described in the previous paragraph, it functions slightly differently in order to avoid finding repeated sequences. The method used here was inspired by Nilsson and Goldberger [2001]. The algorithm starts by finding the minimal sequence, s^1 , using the Viterbi Algorithm, as described above. It then sets up the matrix C , which is expanded by one column during each round, where a round refers to the time spent finding one new sequence. After finding s^1 , the first column of C is created as follows.: We divide the space of all possible sequences, excluding s^1 , into the sets A_i , for $i = 1, \dots, m$:

$$A_1 := \{s \mid s_1 \neq s_1^1\}$$

$$A_k := \{s \mid s_1 = s_1^1, \dots, s_{k-1} = s_{k-1}^1, s_k \neq s_k^1\}, \text{ for } k = 2, \dots, m.$$

Then

$$C(i, 1) := \min_{s \in A_i} \sum_{k=1}^i V_k(s_k) + R_k(s_k).$$

We then find the coordinates of the minimum of C , i.e., let $C(j, i)$ contain the smallest value of C . (At this point, C only has one column, thus $i = 1$, but later it will have more columns, and the system will function in the same way.) Therefore, the next

best minimal sequence, s^2 , is in the set A_j , so

$$s_1^2 = s_1^1, \dots, s_{j-1}^2 = s_{j-1}^1, \text{ and } s_j^2 \neq s_j^1.$$

This gives us the first $j - 1$ values of s^2 , and we can calculate the remaining values using (3.1).

After finding s^2 , we set $C(j, i) := \infty$, and now there should only be potentials larger than the potential of s^2 left in C . We then subdivide the sequence set again, so that it excludes s^2 , as well as s^1 . So we set:

$$A_j := \{s \mid s_1 = s_2^1, \dots, s_{j-1} = s_{j-1}^1, s_j \neq \{s_2^j, s_1^j\}\}$$

$$A_k := \{s \mid s_1 = s_2^2, \dots, s_{k-1} = s_{k-1}^2, s_k \neq s_k^2\}, \text{ for } k = j + 1, \dots, m.$$

Notice that we had to add the precaution to A_j that s_j is unequal to s_2^j and s_1^j , otherwise we would be allowing the set of sequences

$$\{s : s_1 = s_1^1, s_2 = s_2^1, \dots, s_j = s_j^1\}$$

to reenter the set of possible sequences, and s^1 is contained in that set. We then expand C to a $(2 \times m)$ matrix, and set $C(1, 2) = \infty, \dots, C(j - 1, 2) = \infty$, since the sets $A_1, \dots, A_{j-1}$ would contain s^1 and s^2 , and set

$$C(i, 2) := \min_{s \in A_i} \sum_{k=1}^i V_k(s_k) + R_k(s_k).$$

The algorithm continues this way, finding one sequence at a time. It can be summarized as follows:

In step l :

- 1.) Divide the set of all sequences into subsets, which eliminate $s^1, \dots, s^{l-1}$, but contain all other sequences.
- 2.) Find the minimal potential of each of these subsets, and enter it in the appropriate position in C .
- 3.) Find the indices of the minimum of C : $C(j, i)$.
- 4.) Then $s^{l+1} \in \{s \mid s_1 = s_1^i, s_2 = s_2^i, \dots, s_{j-1} = s_{j-1}^i, s_j \neq s_j^i\}$.
- 5.) Set $C(j, i) := \infty$, and find the rest of the sequence s .

This sequence is repeated for each l .

The only difficulty is keeping track of all sequences with the same histories, similarly to how we had to ensure that $s_j \neq \{s_2^j, s_1^j\}$ while defining A_j above. Therefore, to be able to restart this algorithm without repeating calculations, we must record the matrix C , the set of optimal sequences, S , and some information about the locations of past minima of C .

3.4 Phase II: Continuous Optimization

In Phase I, we found possible secondary structure class sequences $s = s_1, \dots, s_{n-3}$. Each secondary structure class sequence gives us bounds for the variables⁴ c , c^+ , and z in $\mathbb{R}$, that is:

$$\begin{aligned} \text{for } i = 1, \dots, (n-3) : \underline{c}_{s_i} &\leq c_i \leq \bar{c}_{s_i}, \\ \underline{c}_{s_i}^+ &\leq c_{i+1} \leq \bar{c}_{s_i}^+, \\ \text{and } \underline{z}_{s_i} &\leq z_i \leq \bar{z}_{s_i} \end{aligned}$$

with $\underline{c}_{s_i}$, $\underline{c}_{s_i}^+$, and $\underline{z}_{s_i} \in \mathbb{R}$.

We can now use the following optimization problem to find the optimal value $\mu_i = \begin{pmatrix} c_i \\ c_{i+1} \\ z_i \end{pmatrix}$ for $i \in \{1, \dots, n-3\}$ and $\underline{z}_i$, for $i \in \{1, \dots, n-3\}$. That is $i \in \{1, \dots, n-2\}$ for c and $i \in \{1, \dots, n-3\}$ for z .

$$\begin{aligned} \min_{\mu_i} & \|R_{\gamma_i, s_i}^{-T}(x_i - \mu_{\gamma_i, s_i})\|^2 + \text{ent}_{\gamma_i, s_i} \\ \text{s.t. } & \underline{c}_{s_i} \leq c_i \leq \bar{c}_{s_i} \\ & \underline{c}_{s_i}^+ \leq c_{i+1} \leq \bar{c}_{s_i}^+ \\ & z_i \leq \bar{z}_{s_i} \\ & \sum_{i=1}^n \sum_{k=1}^{i-1} \frac{1}{\|C_i - C_k\|^2} < \frac{1}{c_{ik}^*} \text{ See 3.4.2 for non-condensed version.} \\ & \underline{H} \leq \sum_{i=1}^n V_{hydro}(x_i, a_i) \leq \bar{H}. \end{aligned} \tag{3.2}$$

⁴these variables are defined in 3.1

Just a quick refreshment on the meaning of the variables and parameters:

The values R_{γ_i, s_i} , μ_{γ_i, s_i} , and ent_{γ_i, s_i} are potential terms determined by data analysis. The number of amino acids is given by n , and the number of 4-fragments is given by $(n - 3)$.

The “primary” sequence class of the i^{th} 4-fragment, $i = 1, \dots, n - 3$, is denoted by $\gamma_i \in \{1, \dots, n_\gamma\}$.

The secondary sequence class of the i^{th} 4-fragment, $i = 1, \dots, n - 3$, which was predicted by discrete optimization before continuous optimization, is denoted by $s_i \in \{1, \dots, n_{sec}\}$. (See 3.1.)

For each of the $(n - 3)$ four-fragments, the geometry values are given by $c_i, i = 1, \dots, n - 2$, $z_i, i = 1, \dots, n - 3$. (See 3.1.)

Upper and lower bounds on the geometry variables c_i and z_i , which depend on the secondary structure class of the fragment i , are given by $\bar{c}_{s_i}, \underline{c}_{s_i}, \bar{c}_{s_i}^+, \underline{c}_{s_i}^+, \bar{z}_{s_i}, \underline{z}_{s_i}$, where $i = 1, \dots, (n - 2)$, for c , and $i = 1, \dots, (n - 3)$, for z .

The coordinates of the C_α atoms are given by $x_i, x_k, i, k \in \{1, \dots, n\}$.

The distance between any two C_α atoms is given by $c_{i,k} = \|x_i - x_k\|, i, k \in \{1, \dots, n\}$.

The amino acid sequence in cartesian coordinates is denoted by $a_i, i = 1, \dots, n$.

The function V_{hydro} is defined as $V_{hydro}(x_i, a_i) = \|R_{hydroa}^{-T}(x_i - \mu_{hydroa})\|^2 + ent_{hydroa}$, where R_{hydroa}^{-T} , $R_{\mu_a}^{-T}$, and ent_{hydroa}^{-T} are potential terms from a covariance model describing the typical location of an amino acid in relation to its protein’s center. (See 3.4.2.)

3.4.1 Description of the Variables and Parameters

The following variables and parameters are used for the protein $p = a_1, a_2, \dots, a_n$ with n amino acids a_i :

The bounds $\bar{c}, \underline{c}, \bar{c}^+$, and $\underline{c}^+$ are upper and lower bounds on c_i and c_{i+1} . Since c_i and c_{i+1} are cosines, the natural bounds are ± 1 . We have been using the actual minimum and maximum of c_i and c_{i+1} for each fragment of type $s \in 1 \dots n_s$ in the data set instead, since they are tighter, and therefore yield a smaller sized search space than the natural bounds. Eventually, however, the more general bounds should be used.

Instead of the natural bounds for z , we have also been using the actual minimum and maximum of z_i for each fragment of type $s \in 1 \dots n_s$ in the data set as $\underline{z}$ and $\bar{z}$. With the HEFG classification, z_i is the sine or the cosine of the torsion angle, and therefore

between $\pm \frac{\sqrt{2}}{2}$. In the original classification of the torsion angle, ϕ_i was based on a tangent, so it could be infinitely large, and therefore the natural bounds on z would be $\pm\infty$. The parametrization let z_i be in an interval with upper bound ≤ 100 and lower bound ≥ -100 .

The secondary structure class sequence $s = \{s_1, \dots, s_{n-3}\}$ is the sequence of the secondary structure class of each 4-fragment. They are also referred to as *sec*-classes, because they describe local (on 4-fragments) geometry structures in proteins, and the local geometry structures determine the secondary structure.

The γ classes⁵, which describe the primary and the secondary sequence, but are referred to as the “primary sequence classes,” were developed by data analysis. If we know the amino acid sequence of a 4-fragment, we can assign it exactly one of the n_γ possible classes. That is

$$\begin{aligned}\gamma : \{4\text{-fragments}\} &\longrightarrow \{1, \dots, n_\gamma\} \\ a_{i:i+3} &\mapsto \gamma_i.\end{aligned}$$

The potential terms R_{γ_i, s_i} , μ_{γ_i, s_i} , and $\text{ent}_{\gamma_i, s_i}$ were determined by a covariance model⁶ describing c_i and z_i .

3.4.2 Description of the Constraints

The close contact constraint is

$$\sum_{i=1}^n \sum_{k=1}^{i-1} \frac{1}{.0001 + \max(\|C_i - C_k\|^2, .0001)} < \frac{1}{c_{ik}^*}.$$

The variables C_i , $i \in 1, \dots, n$ represent the C_α coordinates of the i^{th} amino acid in $\mathbb{R}^3$. The parameter c_{ik}^* is the value of the denominator of the first term calculated for the actual coordinates of the amino acid. Naturally, this parameter would be unknown, if we were making an actual prediction. It therefore could be replaced by a parameter representative of the size of the protein, i.e. the average value of c_{ik}^* for proteins of size n . The value .0001 is added to the denominator, so that the denominator is guaranteed to be unequal to zero. Similarly, we take the maximum of the squared

⁵The γ -classes were developed using a Naive Bayes approach, which can be read about in Hattinger [2008].

⁶This method and its applications are described in Hattinger [2008].

norm and .0001.

The close contact constraint bounds the allotted distance between each two amino acids in the protein. The upper bound on the sum prevents two amino acids from getting too close.

The “centeredness constraint” describes the location of an amino acid in relation to the protein’s center.

For each of the 20 amino acids, the following ratio is computed for each of the r_a occurrences of the amino acid in the data set, $i = 1, \dots, r_a$:

$$C_i(a) = \frac{\|x_i - \bar{x}\|^2}{\chi(n)}$$

The location of the amino acid in cartesian coordinates is given by $x_i \in \mathbb{R}^3$, and the location of the protein’s center is given by $\bar{x}$. The function $\chi(n)$ describes the average distance of an amino acid from the center of a protein with n amino acids. This gives the ratio of the distance of the amino acid from the protein’s center to the average distance of amino acids from the center. Therefore, it indicates if this amino acid is one of the protein’s outer amino acids or one of the protein’s inner amino acids. We save all of the ratios for each amino acid in a $(1 \times r_a)$ vector. The potential terms R_{hydro_a} , μ_{hydro_a} , and ent_{hydro_a} were found for each amino acid a using a covariance model describing each of the 20 vectors. (See Burger.) The “hydro” potential is then:

$$V_{hydro} = \|R_{hydro_a}^{-T}(x_i - \mu_{hydro_a})\|^2 + \text{ent}_{hydro_a}$$

Really R is not transposed, since R is a number, and the norm is just an absolute value. But writing the terms as if they were vectors keeps the formulation consistent with the definition of a covariance model. The *hydro* constraint is:

$$\underline{H} \leq V_{hydro} \leq \overline{H}$$

The upper and lower bounds on V_{hydro} are determined based on the maximal and minimal values of V_{hydro} , calculated with the actual protein structure, of all proteins with length n in our data set.

The intention of this constraint is to encourage structures in which amino acids which are typically near a protein’s center are near the center.

3.4.3 Optimization

The continuous optimization phase minimizes (3.2). We perform the optimization with Knitro, a software program for nonlinear optimization problems, which is discussed in detail below. The quality of the models found through optimization is judged by the GDT_TS score using LGA, so that we know how accurate our models are in comparison to the predictions made at CASP. We use either no initial values, the values of c_i and z_i corresponding to the actual tertiary structure, or the average values of c_i and z_i for each class as the initial values for optimization.

3.5 Software

We use several software products for this project. AMPL is a modeling language used to represent optimization problems in a computer-readable format. For local optimization, we are using Knitro 5.2 [Byrd et al., 2006, Waltz and Plantenga, 2008], which is very flexible and was developed for large nonlinear programs. In order to compare our predictions with CASP predictions, we use the structure comparison program LGA, which was discussed in the CASP section. We used MATLAB for all of the programs we wrote ourselves.

3.5.1 AMPL

AMPL is a modeling language for optimization problems. The convenience of AMPL is that each optimization program requires the user to input the problem in a very specific way. AMPL is able to translate the model into the input format required by each of the products, so that the user only has to understand AMPL, and not all of the other programs. [amp, 2008]

3.5.2 Knitro

Knitro 5.2 is a state of the art solver for local optimization problems. (See the Waltz and Plantenga [2008] or Byrd et al. [2006].) It is suitable for solving large (i.e., 1000 variables and more) nonlinear programs. Knitro gives the user three algorithms to

choose from, and suggests that the user try out all three, because its hard to predict which algorithm will work best. Knitro can also suggest an algorithm, if the user is particularly clueless or lazy. It is also possible to crossover from an interior-point method to an active set method after the interior-point method has finished, to see if the active set method can make an improvement on the solution (or non-solution).

Knitro solves problems of the form:

$$\begin{aligned} \min_{x \in \mathbb{R}^n} & f(x) \\ \text{s.t. } & c^L \leq c(x) \leq c^U \rightarrow \text{bounds on the general constraints} \\ & b^L \leq x \leq b^U \rightarrow \text{simple bounds on the variables} \end{aligned}$$

It assumes that the objective function $f(x)$ and the constraint functions $c(x)$ are smooth, but has ways of dealing with non-smooth functions. The constraint functions $c(x)$ can be divided into a set of equality constraints, $c_E(x)$, and a set of inequality constraints, $c_I(x)$, which can be made into equality constraints using the slack variables s so that $c_I(x) + s = 0$. Knitro's goal is to retrieve a feasible, optimal point. To do this, it uses the following version of the first order Karush-Kuhn-Tucker Optimality Conditions:

$$(1) = \begin{cases} \nabla_x \mathcal{L}(x, \lambda) = \nabla f(x) + \sum_{i=1}^m \lambda_i^c \nabla c_i(x) + \sum_{j=1}^n \lambda_j^b = 0 \\ \lambda_i^c \cdot \min((c_i(x) - c_i^L), (c_i^U - c_i(x))) = 0, i = 1, \dots, m \\ \lambda_j^b \cdot \min((x_j - b_j^L), (b_j^U - x_j)) = 0, j = 1, \dots, n \end{cases}$$

with $\lambda_i^c, \lambda_j^b \leq, \geq$ defined as necessary, depending on finiteness of bounds

$$(2) = \begin{cases} c_i^L \leq c_i(x) \leq c_i^U \\ b_j^L \leq x_j \leq b_j^U \end{cases}$$

The optimality error at the point $\hat{x}$ is determined by the maximal violation of each of the conditions in (1), and the feasibility error is defined by the maximal violation of each of the constraints in (2). A point is considered a locally optimal solution, if both the feasibility error and the optimality error are below user defined/default limits.

3.5.2.1 3 Algorithms

Two of Knitro's Algorithms use Interior Point Methods to find a local optimum. Both of these algorithms solve a series of barrier subproblems, decreasing the barrier parameter after at least one minimization step on the current objective. (See 2.2.4.) The barrier problem solved with parameter μ is:

$$\begin{aligned} \min_{x,s} \quad & f(x) - \mu \sum_{i=1}^m \log s_i \\ \text{s.t.} \quad & c_E(x) = 0 \\ & c_I(x) - s = 0 \end{aligned}$$

The two algorithms differ in what they do after this. The **Interior/Direct** algorithm uses a line search (with Newton's Method) and the **Interior/Conjugate Gradient** algorithm uses a trust region approach for the descent. The **Interior/Direct** algorithm solves the primal-dual matrix using linear algebra, while the **Interior/Conjugate Gradient** method uses a projected conjugate gradient to solve the primal-dual problem. The **Active Set** method is typically better than the Interior Point methods, if starting at a good initial point, so we often start our optimization program with the interior/direct algorithm, and then switch to this method for the final few iterations.

3.6 Implementation

Programs written to carry out the algorithm are stored in

`\virgy\prot\ginny.`

3.6.1 Phase I: Discrete Optimization Program

Running the discrete optimization program is simple. There are a lot of input variables and a lot of output variables, but they do not need to be typed in, if the program is being called from an outside program, which it usually is.

```

function [S,SV,CL,jsofeachs,endj,endi,pastis,CLcol,jeq1,L] =
    onesequenceforp(p,data,confus,class_info,newclasses,nums,nfound,CL,
        jsofeachs,endj,endi,past,CLcol,S,SV,jeq1,L)
% this function should provide the same result as sequencesforp, but should
% work by finding exactly one sequence at a time, starting with the very
% best. (sequencesforp finds the nfound best sequences simultaneously).
% the benefit of this method is that it stores all the info. calculated in
% order to find each sequence, so that if more sequences are desired, it
% does not repeat any calculations. it is based on an article: "Sequentially
% Finding the N-Best List in HMMS"
%
% most one the time, one is not interested in ordering new sequences at a
% later point, and then the output S suffices, which gives the matrix with
% the number of predicted sequences ordered (nfound)
%
% Input:
% p: protein number
% data, confus, class_info: files created in startup.m
% newclasses, nums: also created in startup.m (newclasses = 1 for hefg,
%                                     nums = 4 in that case)
% nfound: number of sequences desired, typically around 20
%
% the remaining input variables only need to be inserted if more sequences
% are desired. they can be returned the first time this function is called,
% and then used as input
%
% Output: S: matrix with nfound sequences (one row per sequence)
%         SV: potentials of each sequence
%         - the remaining output variables need to be returned in the
%           sequence will be called again.
%
% so, this will save the potentials of tons of sequences in a matrix called
% C. really, each entry of C just saves the best potential of all sequences
% in the set  $B(i) \sim \{s | s_{\{1\}} = sb_{\{1\}}, s_{\{2\}} = sb_{\{2\}}, \dots, s_{\{i-1\}} = sb_{\{i-1\}}, s_{\{i\}} \sim sb_{\{i\}}\}$ , where sb is one of the sequences already determined to be the
% nth best. (nth = first, second, third,...)
% then the minimum of C is found, and the sequence corresponding to that

```

```
% potential is derived, partially through its relationship to the sequence
% sb, and partially with the values Rk made by dynopt.
%
% Caution: doesn't seem possible to just order less than 3 more sequences -
% fix that!
```

3.6.2 Phase II: Continuous Optimization Program

The continuous optimization problem is called with the following function. It, too, has lots of input, which is needed to request specific constraints and initial values.

```
function [GDT,LGArmsd,rmse] =
    runopt(data,class_info,newclasses,confus,probnam,p,alnum,maxit,clco,
        hp,initials,avginitials,thfrags,zlu,displaymode,individualopt,s)
%creates the ampl file with the constraints and initial values requested,
%runs the optimization program, and then calls lga to find the gdt_ts and
%rmse values of the prediction. makes pictures as well!
%
%GDT = runopt(data,class_info,newclasses,confus,probnam,p,alnum,maxit,
%    clco,hp,initials,avginitials,thfrags,zlu,displaymode,individualopt,s)
%
% Input:
% data,class_info,confus: data files created by startup.m of the same name
% newclasses: 1 for hefg, 0 otherwise, also fixed in startup (if newclasses
%    is switched, startup should be run again, bc newclasses has an
% effect on several things)
% probnam: typically 'tester', but whatever one desires. just in ' '
% alnum: options 1,2,or 3...
% alnum = 1: Knitro's Interior/Direct Algorithm
% alnum = 2: Knitro's Interior/CG Algorithm
% alnum = 3: Knitro's Active Set Algorithm
% maxit: maximum number of iterations for ampl (default = 10000)
% clco, hp, and thfrags only work for original classes (newclasses = 0)
% clco: use close contacts constraint -- cc specifies amount allotted to be
% hp: uses hydro constraint
% initials: 1, start at actual values of c_i and z_i, 0, don't start at
```

```

% actual values
% avginitials: 1, start at avg values of c_i and z_i for that class, 0
%   don't
% zlu (z lower,upper) = 1: lower and upper bounds for z
%   = 0: only one joint bound for z (abs(z) < zboth)
% displaymode = 0: display less ampl output
% individualopt: used with boundchecker, otherwise set to 0. determines
%   whether usual bounds or individual bounds on c and z are used
% usually don't input s, unless want to try a specific secondary structure
%   class sequence
%   if no class sequence provided, use actual sequence
%
% OUTPUT:
% GDT: GDT_TS score from LGA
% LGArmsd: rmsd score from LGA
% rmse: rmsd score using register

```

3.6.3 Phase I and II Together

To run the entire optimization problem, the function `optprogram.m` is used:

```

function [meanGDT,meanrmse,minGDT,maxrmse,optGDT,optrmse,allpres,GDTs,
    bestsGDT,bestpreGDT,bestsRMSE,bestpreRMSE,S,sactual,bestpreGDTrmse,
    GDTmax,bestpreRMSEGDT,rmsemin,GDTactual,rmseactual]
    = optprogram(p,confus,newclasses,class_info,data,probname,clco,hp)
% finds likely class sequences for the protein p,
% then optimizes with each of the possible sequences,
% and then chooses the sequences with the lowest rmsd and highest GDT
% as the winning sequence
% plots all structures together, and 'winners' as well as prediction using
% actual sequence separately
%
% INPUT:
% input is similar to that of runopt and co, but some parameters are fixed
% in this function, and have to be changed as needed.
% keeping in mind, that clco, hp, and thfrags don't work for hefg :(

```

```
% p: protein predictions are being made for
% confus, class_info, data: data files (of the same name) made by
%                               startup_ginny.m and needed
% newclasses: 1 for hefg, 0 for original
% probname: can be anything, just put in ' ', and shouldn't try too hard to
% be cute
% clco: use close contact constraints
% hp : use hydro constraints
%
% OUTPUT:
% most of the output is just needed for makelotsoftables.m, which
% makes a lot of tables showing the results for various proteins with various
% constraints and initial values - see that program
% interesting outputs are the originals, which are described like this:
% returns rmse of all structures, the sequence (bests) with the best rmse,
% and the ti, ci values of the best sequence (bestpre.c, bestpre.t)
```

This program runs the continuous optimization program for 19 predicted secondary structure class sequence, and the actual sequence.

4 Results

As described in the Section 3, a goal of this project was to find a solution $\mu_i = \begin{pmatrix} c_i \\ c_{i+1} \\ z_i \end{pmatrix}$ of the following optimization problem:

$$\begin{aligned} \min_{x_i} & \|R_{\gamma_i, s_i}^{-T}(x_i - \mu_{\gamma_i, s_i})\|^2 + \text{ent}_{\gamma_i, s_i} \\ \text{s.t. } & \underline{c}_{s_i} \leq c_i \leq \bar{c}_{s_i} \\ & \underline{c}_{s_i}^+ \leq c_{i+1} \leq \bar{c}_{s_i}^+ \\ & z_i \leq |\bar{z}_{s_i}| \end{aligned}$$

and various constraints are fulfilled.

The variables c_i and z_i describe the geometry of the i^{th} 4-fragment of a protein with n amino acids. The optimization program has two phases. The first phase of the optimization algorithm uses the amino acid sequence to predict the *sec*-class sequence (the secondary structure class sequence.) The *sec* class of the i^{th} 4-fragment (a_i, a_{i+1}, a_{i+2} , and a_{i+3}) provides upper and lower bounds for the values of c_i , c_{i+1} , and z_i of that fragment and parameters for the objective function. Since the secondary structure class sequence is already known for the proteins in the test set (see below), it makes sense to see how well the second phase of the algorithm (the continuous optimization problem) predicts the tertiary structure of a protein, if the secondary class sequence is provided. Therefore, in sections 4.1 and 4.2, predictions made by the continuous optimization program with the correct secondary class sequence as input into the program are discussed. Section 4.3 discusses secondary class structure sequence predictions made by Phase I of the algorithm, as well as the tertiary structure predictions made by Phase II using the sequence predictions made in Phase I.

To test the optimization program, a test set was built using a small set of proteins of various sizes from the data set. The set had to remain small, in order to collect

data in a reasonable amount of time. Five proteins were selected randomly from each of the following size ranges: 30 - 60 amino acids, 60 - 90 amino acids, . . . , and 180 - 210 amino acids. One of the proteins selected from the 60 - 90 amino acid side range had an amino acid larger than twenty, so this protein was discarded from the test set, and never replaced. Thus, a total of 29 proteins from the data set were used to test the algorithm. In order to eventually evaluate the algorithm more accurately, proteins from outside of the data set must be selected for the test set. But for now, it suffices to see if the optimization program at least works with protein's from the data set used to create the parameters for the program.

The data set is stored in the file RT071127.mat. It contains a selection of high quality data from the DSSP data base. See Hattinger [2008].

4.1 The Original 27 Classes

Originally, a system of 27 *sec*-classes for the continuous optimization problem was used. This class system was later discarded and replaced with a different classification, but a sample of the results from structure predictions using these classes is presented in this section, so that these predictions may be compared with those made by the HEFG classes, which replaced these.

The following tables show the GDT_TS values¹ and the RMSD values of predictions made using the original class system. These predictions were made with the actual secondary structure sequences, that is, Phase I of the algorithm was left out. Let n be the length of protein p . The continuous optimization problem received the following input:

$s^* = s_1, \dots, s_{n-3}$, the secondary class sequence of the actual structure
 $\gamma = \gamma_1, \dots, \gamma_{n-3}$, the primary class sequence, which can be derived directly from the amino acid sequence

For each protein p , the following procedure was used to test various prediction methods:

Three possible initial value sets were tested for each protein:

First, the actual values of c_i and z_i were used to start the optimization. If actual

¹see Section 1.4.1

predictions were being made, there would be no knowledge of these values before optimization, because they are exactly the values that the optimization problem strives to find. However, starting at the points that the optimizer is looking for is an easy way to see how well the program works when starting with very good initial values. Second, the average values of c_i and z_i for the sc class of the i^{th} 4-fragment were used. For each class s , these averages were taken over all fragments of class s in the entire data set. If actual predictions were being made, these initial values could still be used, because they depend on the data from the test set, and have nothing to do with one protein in particular.

Finally, no initial values were used. That is, Knitro (see section 3.5.2) was allowed to choose the starting values.

For each of the three initial value sets considered, the optimization program was run three times using different sets of constraints each time. The first time, both the constraints on the C_α -atoms and the “hydro” constraints were used, in addition to the bounds on the variables c and z . (See Section sec:cc for a description of these constraints.)

The second time, the constraints on the C_α -atoms were used together with the bound constraints on the variables c and z , but without the “hydro” constraints.

Finally, the optimization program was run using only the bound constraints on the variables.

Performing optimization with various combinations of constraints is a way to evaluate the effectiveness of each constraint.

These predictions were made with the actual bounds on c_i , c_{i+1} , and z_i that were described in Section and depend solely on the class of the fragment i .

After these predictions were made, the bound constraints on c_i and z_i were tightened around the actual values of c_i and z_i , to see if the optimization would be more successful on smaller intervals, and to see by how much the size of the interval needs to be decreased in order to reach specific RMSD and GDT_TS values. This was done three times each for a set of percentages: one time tightening the bounds around both c_i and z_i , one time just tightening the bounds around z_i , and one time just tightening the bounds around c_i . For each case, the bounds were tightened around the interval to each of the following percentages: 50%, 25%, 12.5%, 6.25%, and 3.125% of the original distances between c_i (respectively z_i) and the upper (respectively lower) bounds.

The results for three proteins of various sizes are displayed in Tables 4.1 through 4.3.

Note that the best possible RMSD score is 0, and lower RMSD score are better than higher RMSD scores. GDT_TS values range between 0 and 100, with 100 being the best possible score.

Table 4.1: Protein 1a0i_Y03(7 in the test set) with length $n = 35$

	C_α and “hydro” constraints		only C_α constraints		only bound constraints	
	RMSD	GDT_TS	RMSD	GDT_TS	RMSD	GDT_TS
actual initials	2.16	77.143	1.64	84.286	1.74	84.286
average initials	2.31	75.714	1.63	84.286	1.74	84.286
default initials	2.12	76.429	1.49	86.429	1.74	84.286
c and z by .03125	1.78	80	1.56	85	1.46	84.286
z by .03125	1.08	93.571	.89	95.714	.77	96.429
c by .03125	2.9	69.286	2.27	75	2.27	73.571

Table 4.2: Protein 1a0p_Y01 (9 in the test set) with length $n = 61$

	C_α and “hydro” constraints		only C_α constraints		only bound constraints	
	RMSD	GDT_TS	RMSD	GDT_TS	RMSD	GDT_TS
actual initials	2.81	66.803	2.06	83.197	1.52	87.295
average initials	3.03	63.934	2.91	60.246	1.52	87.295
default initials	3.07	66.393	2.34	75.82	1.52	87.295
c and z by .03125	3.04	61.066	2.85	75.41	1.1	92.623
z by .03125	2.69	66.393	2.64	60.656	.74	96.721
c by .03125	3.18	62.705	2.68	71.721	1.75	80.328

Table 4.3: Protein 1aa0 (14 in the test set) with length $n = 113$

	C_α and “hydro” constraints		only C_α constraints		only bound constraints	
	RMSD	GDT_TS	RMSD	GDT_TS	RMSD	GDT_TS
actual initials	3.58	45.575	2.46	72.174	2.56	70.575
average initials	2.61	35.619	2.65	51.327	2.56	70.575
default initials	2.96	39.381	2.48	71.46	2.56	70.575
c and z by .03125	3.3	64.159	2.16	53.319	2.38	80.088
z by .03125	2.52	48.894	2.85	68.584	1.73	85.398
c by .03125	3.28	52.434	3.78	49.779	2.05	74.558

There were so many (27) classes with the original classification system, that it seemed like some of the classes were specific to a very small percentage of the data set, and therefore could only predict the few fragments used to define that class correctly. The helix class greatly outweighed the other classes in the system, which caused the elements of each of the class sequence predictions in the discrete optimization phase to be predominantly equal to one, the class number representing the helix class. That is, the HMM predicted that most fragments in most proteins were members of helices. For example, a prediction of the *sec*-class sequence of a protein with actual secondary

sequence class “2 6 18 1 3 21 12 6 6 5 6” might have been “2 1 1 1 1 2 1 1 1 1 3.” For these reasons, the original classification was rejected in favor of the HEFG classification, which has only four classes, and was designed with the intention of being less specific and more evenly distributed. [Hattinger, 2008]

4.2 HEFG classes

For the HEFG classes, the optimization program was run without any additional constraints; that is, only with the bound constraints on the variables.

The optimization problem was run with the three different initial value sets introduced in Section 4.1. However, the RMSD and GDT_TS values of the predictions found by starting at the actual values of c_i and z_i and by starting at the average values of c_i and z_i for their corresponding secondary structure class were identical to the GDT_TS and RMSD values of the predictions starting at Knitro’s default initial values. This lead to the assumption that the same minimum was found from each starting point, because otherwise it would have been unusually coincidental if the RMSD had been the same repeatedly for two different structures, when it is usually different for different models of the same template. Because of this, only the data found using Knitro’s default initial values will be displayed.

Predictions were made for five to six individual proteins in each size range. The best, worst, and average GDT and RMSD scores for each size range are displayed in tables 4.4 through 4.9. These predictions were made using only the bound constraints on the variables, and no additional constraints. Note that these predictions were the same, regardless of which initial values were used. The protein label in the Protein Data Base and the protein class according to the Protein Data Base are given along with the highest and lowest scores, in order to introduce some members of the test set. Protein names of the form j_Yi refer to the i^{th} domain of the protein of name j . For example, $1ak5_Y04$ refers to the 4^{th} domain of the protein $1ak5$.

From Tables 4.4 through 4.9, it can be seen that the GDT_TS scores decrease and the RMSD increases as the number of amino acids increases.

Using no initial variables (well, Knitro’s default initial values,) the bounds on the variables were tightened by the following percents: 50%, 25%, 12.5%, 6.25%, and

Table 4.4: Proteins of Length $n = 30$ through $n = 60$

	score	protein name in DSSP	protein ID in the test set	protein class
best GDT_TS	87.5	1ak5_Y04	85	mainly α
best RMSD	1.38	1a0i_Y03	7,85	7: loop
average GDT_TS	81.86			
average RMSD	1.79			
worst GDT_TS	70.41	1apf	118	mainly β
worst RMSD	2.65	1apf	118	mainly β

Table 4.5: Proteins of Length $n = 60$ through $n = 90$

	score	protein name in DSSP	protein ID in the test set	protein class
best GDT_TS	77.37	1apj	119	α, β
best RMSD	2.51	1aop_Y01	9	mainly α
average GDT_TS	67.73			
average RMSD	3.07			
worst GDT_TS	51.25	1bd0	207	all β
worst RMSD	3.78	1bd0	207	all β

3.125%, analogously to the method used in Section 4.1 for the original classes. The general pattern of change in RMSD due to different levels of tightening was the same for all the proteins, so only the data from one protein is displayed in 4.10.

In Table 4.10, it can be seen that the RMSD decreases as the bounds are tightened, which is to be expected. In a few cases, which are not seen in this table, the RMSD increased slightly, even though the bounds were tightened. This is probably because the optimization program found a different minimum, having started at the different starting values caused by the tightened bounds. In all cases, tightening the bounds on c_i and z_i together had more effect than just tightening the bounds on c_i or the bounds on z_i . The final RMSD after tightening only the feasible region of c_i to .3125% of the original region was less than the final RMSD (and therefore indicated a more accurate prediction) after tightening only the feasible region of z_i to .3125% in all but one or two of these trials. Therefore, it is more effective to reduce the feasible region of z_i , which represents the torsion angle of the i^{th} fragment, rather than the region around c_i , which represent the interior cosine angles of that fragment, in order to arrive at a better prediction.

Table 4.6: Proteins of Length $n = 90$ through $n = 120$

	score	protein name in DSSP	protein ID in the test set	protein class
best GDT_TS	65.49	1aa0	14	mainly α
best RMSD	2.45	1aa0	14	mainly α
average GDT_TS	59.91			
average RMSD	2.96			
worst GDT_TS	3.41	1be0	209	all α
worst RMSD	55.83	1anv_Y01	104	104: mainly α

Table 4.7: Proteins of Length $n = 120$ through $n = 150$

	score	protein name in DSSP	protein ID in the test set	protein class
best GDT_TS	62.40	1anv_Y05	105	α, β
best RMSD	3.01	1anv_Y05	105	α, β
average GDT_TS	52.20			
average RMSD	3.28			
worst GDT_TS	45.96	1awe	166	mainly β
worst RMSD	3.53	1awe	166	mainly β

4.2.1 Comparison of Original Classes with HEFG classes

Very few data was produced with the original classes, so it is impossible to make any general assumptions. However, it can be seen that (in the four cases tested) the HEFG classes and the original classes provided approximately equally good predictions, but the original data usually provided slightly better predictions. However, one of the problems with the original data was that it seemed overly fitted to the data set. It would be interesting to see how well both classifications perform on proteins outside of the data set.

4.3 Secondary Structure Sequence Predictions

In Phase I, we used a Hidden Markov Model to predict the most likely secondary structure class sequence, given the primary structure class sequence.

For each protein in the test set, the set S of the nineteen most likely secondary class sequences $s = s_1, \dots, s_{n-3}$ was predicted in Phase I of the algorithm. For each of these sequences, the continuous optimization problem in Phase II was solved using no constraints other than the bound constraints on c_i and z_i . The initial values chosen for the optimization problem were the average values of c_i and z_i for the sc -class of

Table 4.8: Proteins of Length $n = 150$ through $n = 180$

	score	protein name in DSSP	protein ID in the test set	protein class
best GDT_TS	62.73	1ble	228	α, β
best RMSD	2.82, 2.83	1bla, 1ass	228, 139	both: α, β
average GDT_TS	47.83			
average RMSD	3.15			
worst GDT_TS	34.71	1aac_A	109	mainly β
worst RMSD	3.61	1afv_A	54	mainly α

Table 4.9: Proteins of Length $n = 180$ through $n = 210$

	score	protein name in DSSP	protein ID in the set	protein class
best GDT_TS	59.72	1ais_B	73	mainly α
best RMSD	2.4	1ais_B	73	mainly α
average GDT_TS	45.63			
average RMSD	3.32			
worst GDT_TS	38	1atl_A	144	α, β
worst RMSD	3.99	1azs_A	181	α, β

the fragment i . That is, the average value of c was found for all fragments in class s and that was taken as the initial value for all c_i in fragments of class s . The same was done for z_i .

For each sequence $s \in S$, a solution of the local optimization problem using secondary structure sequence s was then found, and the GDT_TS and RMSD values were stored.

In Table 4.12, the minimal, maximal, and mean GDT_TS and RMSD values of predictions made with each of the twenty sequences in the set S , for each of the proteins with between 150 and 180 amino acids in the test set, are displayed. The GDT_TS and RMSD values of the prediction using the actual secondary structure sequence s^* are displayed, as well. The GDT_TS and RMSD values of the first sequence in set S are also shown, because the Hidden Markov Model predicted the sequence s^1 to be the class sequence which should best describe the protein's secondary structure.

It is to be expected that the predictions made with predicted secondary sequence classes are not as accurate as predictions made using the actual secondary sequence classes, since the values of the variables c_i and z_i are bounded properly, if the true sc -classes of the fragments are used to make the bounds. It is to be hoped that the predictions made with the first sequence in S provide the best GDT_TS and RMSD scores of all the predictions made with sequences in S .

Notice in Table 4.12 that the maximal predicted GDT_TS score for $p = 109$ is actually higher than the GDT_TS score of the tertiary structure prediction made with the ac-

Table 4.10: Change in RMSD and GDT_TS due to Bound Tightening for Protein 165 in the Test Set (length $n = 94$)

	RMSD	GDT_TS
no tightening	3.08	56.383
c and z by 0.5	3.11	57.713
c and z by 0.25	2.78	64.628
c and z by 0.125	2.12	77.394
c and z by 0.0625	1.52	89.362
c and z by 0.003125	.81	96.543
z by 0.5	3.11	57.713
z by 0.25	2.76	62.500
z by 0.125	2.66	64.894
z by 0.0625	2.49	68.883
z by 0.003125	2.46	70.213
c by 0.5	3.08	56.383
c by 0.25	3.00	59.043
c by 0.125	2.85	63.830
c by 0.0625	2.77	66.223
c by 0.003125	2.74	65.957

Table 4.11: Proteins of Length $n = 180$ through $n = 210$

	score	protein name in DSSP	protein ID in our data set	protein class
best GDT_TS	59.72	1ais_B	73	mainly α
best RMSD	2.4	1ais_B	73	mainly α
average GDT_TS	45.63			
average RMSD	3.32			
worst GDT_TS	38	1atl_A	144	α, β
worst RMSD	3.99	1azs_A	181	α, β

tual secondary structure sequence. This is the only occurrence of a higher GDT_TS for a predicted secondary structure sequence than for the actual secondary structure sequence in the entire test set of 29 proteins. Generally, the maximal predicted GDT_TS score is at least 10 points less than the GDT_TS score of the prediction found using the actual secondary structure sequence.

The **best** RMSD of all nineteen predictions using predicted secondary sequences is smaller (thus better) than the RMSD of the prediction using the actual secondary structure sequence approximately 62% of the time. However, the mean RMSD of all nineteen predictions using predicted secondary sequences is smaller than the RMSD of the prediction using the actual secondary structure sequence only around 38% of the time.

Ideally, the first sequence s^1 in the set S should always provide the best model for the

Table 4.12: Proteins with between 150 and 180 amino acids

protein in test set:	$p = 228$	$p = 172$	$p = 139$	$p = 109$	$p = 54$
size of protein:	$n = 161$	$n = 158$	$n = 152$	$n = 175$	$n = 151$
Actual GDT		43.513	49.342	34.71	48.841
Maximum GDT		33.861	39.309	38.43	44.536
Mean GDT		28.2812	36.23	33.04	37.4649
Minimum GDT		24.525	31.09	29.43	33.278
First GDT		27.057	37.01	33.29	35.762
Actual RMSD		3.57	2.83	2.91	3.61
Minimal RMSD		2.79	2.91	3.00	3.29
Mean RMSD		3.12	3.49	3.28	3.57
Maximal RMSD		3.52	3.80	3.67	3.76
First RMSD		3.08	3.56	3.27	3.61

protein, that is, it should always have the best GDT_TS score in comparison to the other members of the set S . However, that is not the case, as can be seen in Table 4.12. In the test set, the GDT_TS score using $s^1 \in S$ is at least better than the mean GDT_TS score using sequences in S only approximately 28% of the time. The RMSD score is better than the mean around 48% of the time.

5 Conclusions and Outlook

5.0.1 Local or Global Optimization

The structure predictions using the HEFG classes showed no variation in GDT or RMSD when starting at different starting values, so it can be assumed that exactly the same structure was found, regardless of where the optimization started. The HEFG predictions were made using only bound constraints on the variables, and no additional constraints.

Using the original classes and only bound constraints on the variables, there was also no difference in GDT or RMSD when using different initial values. However, there were differences when using the original classes with additional constraints. It would be interesting to see if additional constraints also have an effect of the relevance of the starting point using the HEFG classification. It would also be wise to check the values found using a different initial value option, to see if that has an effect on the structure found. For example, the midpoint between the upper and lower bounds for c_i and z_i could also be used as a starting point.

If it turns out that the initial values have no significant effect on the predicted structure, then it can be assumed that there is only one local minimum in the feasible set, and local optimization should suffice for predictions.

5.0.2 Predictions with the HEFG Classification

The following two figures are taken from Jauch et al. [2007] and show the median and then the best GDT_TS values of the predictions at CASP6 and CASP7.

The next figure shows our predictions, using Phase II of the algorithm with the actual secondary structure sequence.

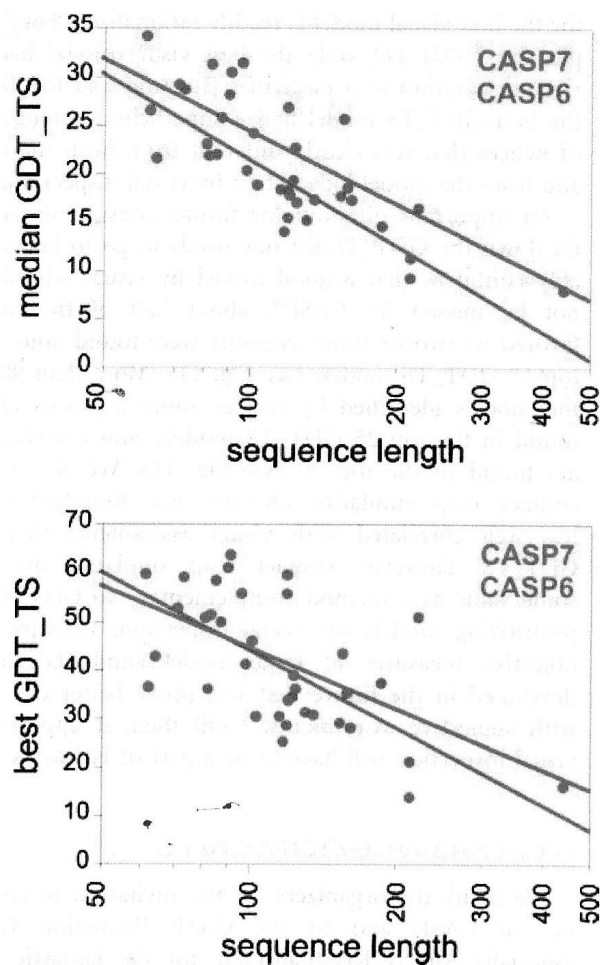


Figure 5.1: Mean and Median GDT_TS Scores from CASP6 and CASP7

While using the actual secondary structure sequence as input for the optimization program, the HEFG classes were so good without any additional constraints. Therefore, it may not be necessary to employ additional constraints during the continuous optimization phase, in order to have a competitive protein structure prediction algorithm. However, to prediction highly accurate structures, the algorithm still needs much improvement, and additional constraints would probably be needed.

The algorithm should be tested for proteins that were not used in the data set, because the parameters used for the optimization that were created by the data are obviously biased toward fragments in the data set.

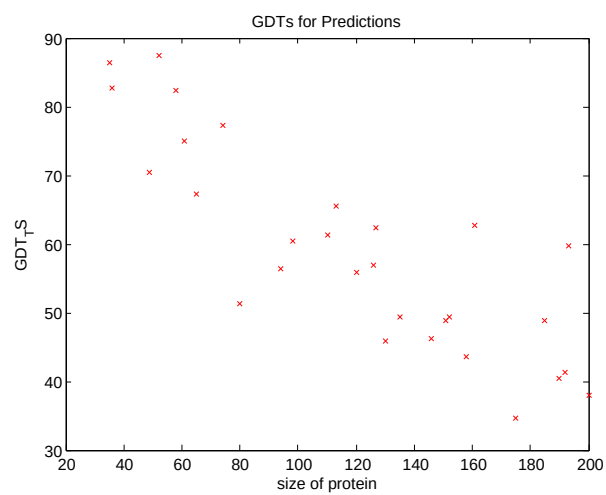


Figure 5.2: GDT_TS Scores of Predictions Using Correct Secondary Structure Sequence

5.0.3 Prediction of Secondary Structure Class Sequence

In the next figure, the mean values of the GDT_TS scores of each prediction, using the sequences from the set S created in Phase I of the algorithm, for proteins of various lengths from the test set are shown.

When compared with figure 5.1, it can be seen that the predictions made by our algorithm are not bad. The next figure shows the best prediction, according to the GDT_TS score, out of the sequences in the set S for the same proteins as above.

These predictions are substantially better than the average predictions. Therefore, Phase I of the algorithm needs to be improved, so that it can distinguish which sequence in S will provide the best prediction.

5.1 Proposed Phase III of the Algorithm

The current algorithm should be followed by a third phase, which would attempt to improve the model. After examining the results of many predictions, global features should be added to the algorithm which will eliminate “bad” structures and encourage “good” structures. Such a feature could be an additional constraint in the continuous optimization program, which would make certain characteristics of structures known to be very unlikely infeasible. Another such feature could be implemented between Phase I and Phase II, to eliminate secondary sequence predictions known to produce poor quality models.

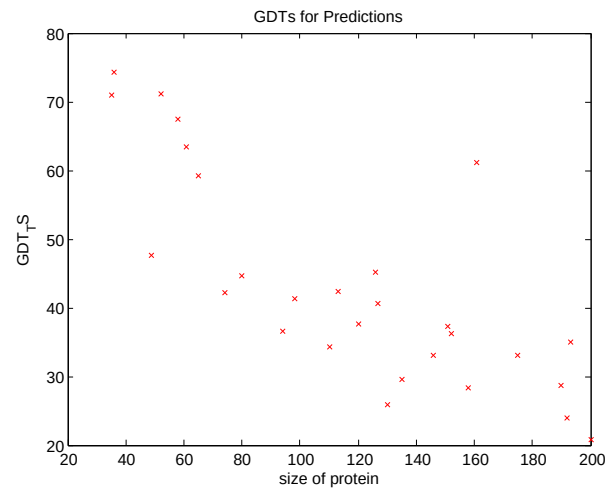


Figure 5.3: Mean GDT_TS Scores from our Predictions

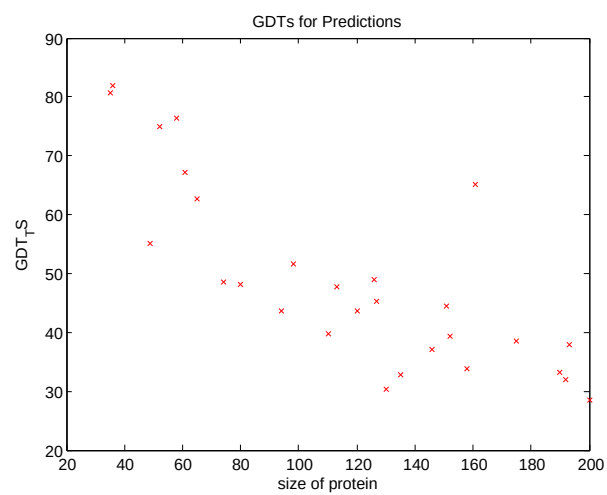


Figure 5.4: Best GDT_TS Scores from from our Predictions

Bibliography

- Ampl a modeling language for mathematical programming, 2008. Available on-line at: www.ampl.com. [58]
- blast website <http://blast.ncbi.nlm.nih.gov/Blast.cgi>. [16]
- foldit solve puzzles for science. interactive online game found at http://fold.it/portal/adobe_main. [13]
- Virginia M. Burger. Data analysis for protein folding - global geometry of proteins. unpublished, contact ginny_burger@hotmail.com. [57]
- Richard Byrd, Jorge Nocedal, and Richard Waltz. Knitro: An integrated package for nonlinear optimization. 2006. Retrieved 29-02-2008 from www.ece.northwestern.edu/tilderwaltz/articles/knitropaper.pdf. [58]
- A. A. Canutescu, A. A. Shelenkov, and Jr. R. L. Dunbrack. A graph theory algorithm for rapid protein side-chain prediction. *Protein Science*, 12, 2003. Retrieved 24-07-2008 from <http://dunbrack.fccc.edu/scwrl3protsci.pdf>. [22]
- Protein Structure Prediction Center. 8th community wide experiment on the critical assessment of techniques for protein structure prediction, 2008. Retrieved April 17, 2008 from <http://www.predictioncenter.org/casp8>. [16, 18]
- Mark Craven. Introduction to protein structure prediction, 2007. lecture notes retrieved from <http://www.biostat.wisc.edu/bmi576/lectures/protein-structure.pdf>. [3]
- George B. Dantzig. Linear programming. *Operations Research*, 50(1). [27]
- Rhiju Das, Bin Qian, Srivatsan Raman, Robert Vernon, James Thompson, Philip Bradley, Sagar Khare, Michael D. Tyka, Divya Bhat, Dylan Chivian, David E. Kim, William H. Sheffler, Lars Malmstro Andrew M. Wollacott, Chu Wang, Ingemar Andre, and David Baker. Structure prediction for casp7 targets using extensive all-atom

- refinement with rosetta@home. Proteins: Structure, Function, and Bioinformatics, 69(S8):118–128, 2007. [21, 22]
- William Elliott and Daphne Elliott. Biochemistry and Molecular Biology. Oxford University Press, second edition, 2002. [3, 4, 6, 7]
- Stephan Hattinger. Datenanalyse fuer proteinfaltung, 2008. unpublished diplomarbeit expected to arrive in the near future on the University of Vienna Department of Mathematics. [43, 45, 56, 66, 69]
- Ralf Jauch, Hock Chuan Yeo, Prasanna R. Kolatkar, and Neil D. Clarke. Assessment of casp7 structure predictions for template free targets. Proteins: Structure, Function, and Bioinformatics, 69(S8):57–67, 2007. [17, 18, 19, 20, 21, 23, 75]
- Steuard Jensen. An introduction to lagrange multipliers. online tutorial, 2008. Available on-line at:<http://www.slimy.com/~steuard/teaching/tutorials/Lagrange.html>. [29, 30]
- Neil Jones and Pavel Pevzner. An Introduction to Bioinformatics Algorithms. MIT Press, 2004. [38, 43]
- Andriy Kryshchak, Krzysztof Fidelis, and John Moult. Progress from casp6 to casp7. Proteins: Structure, Function, and Bioinformatics, 69(S8):3–9, 2007. [23]
- Andrew Moore. Hidden markov models. Tutorial slides retrieved April 11, 2008 from <http://www.autonlab.org/tutorials/>. [38, 41]
- John Moult, Krzysztof Fidelis, Andriy Kryshchak, Burkhard Rost, Tim Hubbard, and Anna Tramontano. Critical assessment of methods of protein structure prediction-round vii. Proteins: Structure, Function, and Bioinformatics, 69(S8):3–9, 2007. [16, 17]
- A. Neumaier. Optimization – theory and algorithms. Manuscript, 2007. Available on-line at:<http://www.mat.univie.ac.at/~neum/opt07.pdf>. [25]
- Arnold Neumaier. Molecular modeling of proteins and mathematical prediction of protein structure. SIAM Review, 39:407–460, 1997. [9, 10]
- Dennis Nilsson and Jacob Goldberger. Sequentially finding the n-best list in hidden markov models. Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), 2001. [52]
- Vijay Pande and Stanford University. Folding@home distributed computing. website: <http://folding.stanford.edu>. [13]

- Lawrence Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. Proceedings of the IEEE, 77(2), 1989. Retrieved on 16-05-2008 from www.cs.ubc.ca/tildemurphy/Bayes/rabiner.pdf. [38, 41, 42]
- Piotr Rotkiewicz and Jeffrey Skolnick. Fast procedure for reconstruction of full-atom protein models from reduced representations. Journal of Computational Chemistry, 29(9), 2008. Retrieved 24-07-2008 from <http://www3.interscience.wiley.com/journal/117885015/>. [22]
- H. Schichl. Numerische mathematik 2. Manuscript, 2001. Available on-line at:<http://www.mat.univie.ac.at/Vorlesungen/ss01/35.html>. [44]
- Richard Waltz and Todd Plantenga. Knitro Users Manual. Ziena Optimization, 2008. [58]
- Adam Zemla. Lga: a method for finding 3d similarities in protein structures. Nucleic Acids Research, 21(13), 2003. [18, 19]
- Adam Zemla. Lga (local-global alignment) a method for finding 3d similarities in protein structures, 2008. Retrieved April 17, 2008 from <http://predictioncenter.llnl.gov/local/lga/>. [19]
- Yang Zhang. Template-based modeling and free modeling by i-tasser in casp7. Proteins: Structure, Function, and Bioinformatics, 69(S8):108–117, 2007. [22]

Resume

NAME	Virginia M. Burger
BIRTHDATE	February 29, 1984
BIRTH PLACE	New Haven, Connecticut

Education

<i>University of Vienna</i>	2003 - Present
Diplomstudium Mathematik	

<i>University of Vienna</i>	2002 - 2003
Studium Biologie	

<i>William G. Enloe High School</i>	1998 - 2002
Raleigh, North Carolina	

Work Experience

<i>Peek und Cloppenburg</i>	sales-girl
Vienna, Austria	October, 2006 - April, 2007

<i>Ernie's</i>	breakfast waitress
Block Island, Rhode Island	Summer 2001 - Summer 2007

<i>The Ice Cream Place</i>	server
Block Island, Rhode Island	Block Island, Rhode Island