Parallel Global Optimization of Proteins

Master's thesis — August 25, 1994

André van der Hoek

Thesis advisors:

Prof. R.B. Schnabel
University of Colorado at Boulder

Dr. H.W.J.M. Trienekens
Erasmus Universiteit Rotterdam

Foreword

The convention says that a foreword can be at most one page. One page? I think one should be able to write a foreword that is longer. First of all, if one has a lot to say, there should be room to do so. Second, I don't like conventions. Third and last, what's in a page? It would be left —intentionally— blank anyway. However, convention is convention. So, despite all the words I have to say, I will try not to fill more than one page.

This thesis is the result of one of the most interesting experiences in my life: working full time as a Professional Research Assistant at the University of Colorado at Boulder, in the mean time being a student at the Erasmus University Rotterdam. It has been quite an adventure to finish classes in Rotterdam, while physically being located in Boulder. I thank Martin van Wijngaarden for our smooth cooperation in this (i.e., him staying up late). I will remember our pleasant talk-sessions, which were a recurring event on my weekly schedule.

Of course this experience would not have been possible without Bobby Schnabel giving me the opportunity to work with him and Richard Byrd in the very interesting area of global optimization. Besides having learned a lot —all of which I am entitled to forget now—, I will mostly remember the pleasant cooperation in our group. Thanks Betty Eskow and Chung-Shang Shao, you were very much part of it.

Harry Trienekens is the person who initiated contact with the University of Colorado after he had told me how great life in Boulder could be. I thoroughly thank him for that. I also want to thank him for being my advisor for this thesis. I know that must have been very hard —yes you will get a beer, maybe even two—, since it was his first time.

I thank Shabnam Erfani for two reasons: reading draft versions of my thesis (and commenting those with great precision) and uhm....., well she knows why.

Another important aspect of my stay in Boulder, that can not go by unnoticed, is *xpilot*. This great network game helped me go through the long hours of writing this thesis. Thanks to Bjørn Stabell, Ken Ronny Schouten and, Bert Gÿsbers for developing this game. I hope many future players will enjoy the game as much as I do. Thanks Lardbottom, Caffeine Freak, TIMID, Roma, Cool, Blade, Schikore, Q, Dr. Death, Ender, PerlPilot, Jmj, PmPilot, Ugly Stick, Jvp, Your mother, I Kill You, Kent, Uncle Fester, Armadillo, Captain Chaos, Phoenix, Swervedriver, Jmartin, and all the other players for a great time. I look forward to meeting you again over the internet to play some of Blood's Music, TEAMBALL, or Corroded Treasures!

Then, of course, many people stayed home when I left for Boulder. First and foremost mom, dad, and sis: thanks for supporting me on my stay here in Boulder. You all made use of me by visiting and staying at my place, but I will forgive you that.

Besides my family, a lot of people supported me by writing tons of mail and sending millions of postcards. Special thanks goes to Jan & Corry Schippers, who are the absolute champions in this profession. The Dutch Postmaster must *love* them!

Finally I would like to say to all the mail and postcard senders: keep up the good work next year!

André van der Hoek

Contents

Chapter 1

Introduction

This thesis is concerned with the development of a large scale global optimization algorithm to solve the *protein folding problem*. This is a very challenging problem and although research in the past decades has resulted in a better understanding of the problem, a solution has not been found yet. As opposed to most research projects in this field, our interest in the protein folding problem stems from a background in numerical computation: we use it as a test application for our recently designed large scale global optimization methodology. This approach is not specifically designed for the protein folding problem, and is applicable to other problems as well. However, in this thesis we will limit ourselves to the application of the approach to the protein folding problem.

The protein folding problem is to predict the unique three-dimensional folded structure of a protein from its primary structure, that is the sequence of amino acids by which the protein is defined. It is one of the fundamental problems in biophysical sciences, and understanding the physics of protein conformations will be of great importance for biomedicine: in designing novel proteins, in decoding the genetic information obtained by the Human Genome Project, in designing new drugs, and in trying to understand the structures and functions of the thousands of proteins that are being discovered every day in biotechnology labs [CD93].

Currently, the *native state* of a protein (i.e., the unique three-dimensional folded structure) can not be predicted from its primary structure. Although biophysicists are able to build new proteins, i.e., build a sequence of amino-acids, a newly constructed protein immediately folds into its native state. For all of the areas mentioned above, this native state is far more important than the primary, unfolded structure. This is because the properties of the folded protein determine its behavior, not the properties of the unfolded protein.

Currently, a protein has to be physically built first, after which its folded state and its properties can be determined using X-ray crystallograhic technology or nuclear magnetic resonance data. This is a trial and error approach, with high cost and great time investment for each trial. Usually it takes many trials —and consequently high cost— to develop one single protein with the desired properties. Clearly, a better and less costly approach is needed.

In this thesis we explore one possible, relatively inexpensive approach to predict the folded state of a protein. We use a well-known empirical energy function to model the state of a protein, and then design a new large scale global optimization algorithm to find the global minimum of this function. The state of the protein corresponding to this global minimum is believed to be the native state we are looking for [Aba93]. This approach is much cheaper than the laboratory approaches, since it does not require any expensive technology. Instead, it *simulates* the folding process. It is also much faster, so many more trials can be done. So far, the new method has been tried on a few relatively small proteins and it has produced the correct results. Research is still going on to determine how well the method performs on larger proteins and what extensions should be made in order to make it more efficient.

Chapter 2 gives some background for this thesis. We explain how global optimization comes into play in our solution to the protein folding problem, describe the potential energy function in detail, and give an exact definition of the problem we are trying to solve. In chapter 3 we highlight some

other approaches to solve the protein folding problem. Chapter 4 presents a detailed description of our global optimization method, and deals with the parallel implementation of this method. In chapter 5 we discuss our computational results obtained so far, and in the last chapter, 6, we set out directions for future research to enhance the method to be able to solve the protein folding problem for real-world proteins.

Chapter 2

Background

2.1 Global Optimization

Throughout the years, a lot of research effort has been put into the problem of *local optimization*. Given a nonlinear function, f, of n variables, this problem is defined as finding a point $\overline{x}$ for which $f(\overline{x}) \leq f(x)$ for all x in an open neighbourhood containing $\overline{x}$. This point is called a local minimizer. Many algorithms have been developed to solve this problem (see [DS83]).

However, a nonlinear function can have many local minimizers in its domain D. Finding the lowest minimizer among these local minimizers is the objective of *global optimization*. This has proven to be a much harder problem than that of local optimization, and research towards solutions of this problem has only recently begun. Many approaches have been proposed, but in general they are only reliable for small-scale problems (i.e., 1-6 variables), whereas the types of problems we are interested in have from dozens to thousands of variables, depending upon the protein size and the parameterization.

We do not include a general review of global optimization in this thesis; for recent reviews, see [KT89] or [PSX94]. However, in chapter 4 we do review the approach that is most relevant to our research in protein folding.

2.2 Cluster Problems

A class of molecular chemistry problems that has some relation to protein folding, and that has been considered by a number of global optimization researchers, is the class of *cluster problems*. In this class of problems, one is concerned with finding *the native state* of a cluster of atoms or molecules. This state is the unique equilibrium state in which the cluster occurs in nature at the lowest temperature possible (0 K). It is widely believed that in this native state, the *potential energy function* of the cluster is globally minimized. The potential energy function describes the interactions among the atoms or molecules in the cluster. Finding the exact potential energy function is a very hard problem, therefore empirical potential energy functions are used, which give a sufficient approximation of the exact potential energy. Given such an empirical potential energy function, the problem of finding the native state of a cluster is a global optimization problem with this function as the objective function.

Examples of cluster problems are the Lennard-Jones and the Water problems. The Lennard-Jones problem is to find the native state of a group of identical atoms, whose potential energy is given by the sum of pairwise Lennard-Jones potentials. Global optimization approaches to solve this problem have been explored by many researchers, including [BES92], [Xue94], and [Nor87]. The Water problem is to find the native state of a group of water molecules, whose potential energy is given by a sum of energy terms between pairs of water molecules. A global optimization approach to this problem has been described in [BDE+93] and [Old93].

Characteristic of both the Lennard-Jones and the Water problem is that their potential energy functions have lots of local minimizers, many near the global minimum. Therefore, finding the global minimum, and consequently the native state, is a very hard problem.

2.3 The Protein Folding Problem

A problem that is closely related to the cluster problems mentioned above, is the problem that will be addressed in this thesis: the *protein folding problem* [CD93] [Ric91]. The protein folding problem is defined in [Anf73] as the problem of finding the native state of a protein in its normal physiological milieu. As in the cluster problems, this is believed to be a unique state. Before we exactly define the problem in terms of global optimization, we first give a brief introduction in the chemistry of proteins, followed by the terminology used in this thesis, after which we give a description of the potential energy function used.

2.3.1 Chemistry

A protein is a sequence of amino acids linked together in a chain. In nature twenty different amino acids exist, which all have the same basic structure: a backbone and a side chain. The backbone is the same for each amino acid, whereas each of the amino acids has a unique side chain. The amino acid sequence of a protein is called its *primary structure*. Remarkably, this primary structure defines the native or folded state of the protein completely [Ric91]. However, the rules according to which the protein folds are not discovered yet, and are often referred to as 'the second half of the genetic code', which implies that the problem of finding these rules is a very hard problem. In identifying the folded state of a protein, one usually recognizes an intermediate state first: *the secondary structure*. Looking at the final, folded state —or *tertiary structure*— of the protein, one can identify regions which have distinct characteristic shapes. Such a region is said to have secondary structure. Three characteristic shapes exist:

α-helices the backbone atoms of the amino acids in the region of secondary structure form a helix;

β-sheets the backbone atoms of the amino acids in the region of secondary structure form two or more adjacent strands running parallel or antiparallel (parallel, but strands go in the opposite direction);

random coil the backbone atoms of the amino acids in the region of secondary structure do not assume any regular shape.

The tertiary structure of the protein then can be seen as assembled of individual pieces that have such secondary structure. This tertiary structure is the final folded state in which the protein resides. It is believed that a protein first folds some regions into secondary structure, after which it folds these regions into its final tertiary structure [Bal89]. Research is still going on in determining whether a protein folds through a single sequential pathway [Cre88], or whether there is no unique folding pathway [SK90]. It has even been proposed that a protein folds like humans solve a jigsaw puzzle, with multiple routes to a single solution [HD85]. However, consensus exists on the hypothesis that there is a single folded state that will occur at the end of the folding pathway.

2.3.2 Terminology

Besides the notion of primary, secondary, and tertiary structure, it is useful to define some other terms that will be used frequently throughout this thesis. We start by describing two different, fully equivalent coordinate systems that are used to describe the conformation of a protein: internal and external (or cartesian) coordinates. The position of a group of atoms in space can be specified in either one of these representations. In external coordinates, we explicitly represent the cartesian

coordinates of each atom (i.e., the x, y, and z coordinate of each atom). An alternative approach is to use internal coordinates, which are more closely related to the structure of a protein. In internal coordinates, we use the *bondlength*, *bondangle*, and *dihedral angle* notions to specify the coordinates of the atoms. The bondlength is simply defined as being the Euclidian distance between two consecutive atoms. The bondangle is the angle between three consecutive atoms, assuming the atoms are located in a single plane.

Just the bondlength and the bondangle are not sufficient to specify the position of an atom in a protein chain; its position still has one degree of freedom left. Given a sequence of four atoms, the first and fourth atom can still rotate around the axis of the middle two. The notion of a dihedral angle comes into play here. In the sequence of four atoms, it is the angle between the plane defined by the first three atoms in the sequence and the last three atoms in the sequence (so the middle two atoms are part of both planes). Given the position of the first three atoms in the sequence, the bondlength between the last two atoms, the bondangle formed by the last three, and the dihedral angle formed by all four, the position of the fourth atom is completely determined. We call this kind of dihedral angle a *proper dihedral angle*.

Another case occurs when one considers side chains: a sequence of three atoms with a fourth atom bonded to the second. In this case, we can not use the proper dihedral angle. Instead, we define the so-called *improper dihedral angle* as being the angle between the plane formed by the three atoms in sequence and the plane formed by the first and third atom in the sequence, together with the fourth atom.

These two variants of dihedral angles are sufficient to specify the position of every atom in the protein, given the position of the first three atoms in the whole sequence of atoms in the complete protein.

Note that the number of variables in both the internal and external representation are the same; therefore there is no advantage in using either one from this perspective. However, other reasons influence the choice of representation. For example, it is easier to keep bondlengths and bondangles fixed in internal coordinates than in external coordinates. This will reduce the dimensionality of the problem to 1/10-th of the original dimensionality, without the protein losing its ability to fold itself into the native state. Furthermore, one of the three dihedral angles in the backbone of an amino acid is rigid. Therefore, we are able to further reduce the number of free variables per amino acid to two: the two free dihedral angles in the backbone of the amino acid. This gives the internal representation —fixing most of the free variables— an advantage in dimensionality of 1/15-th over the external representation.

2.3.3 The Potential Energy Function

As in molecular cluster problems, it is widely believed that in the native state of a protein the potential energy of the protein is globally minimized. Therefore, we need a *potential energy function* to model the energy of a protein. Many potential energy functions have been developed to model proteins. Among these, perhaps the three most widely used are ECEPP, AMBER, and CHARMM. ECEPP has been developed and improved by Scheraga and co-workers [MMBS75] [NPS83]. It differs from AMBER and CHARMM in the sense that a lot of the terms that can vary in these two energy functions, are rigid in ECEPP. AMBER, developed by Weiner et al [WKNC86], and CHARMM, developed by Brooks et al [BBO$^+$83], are very similar with only slight differences in the energy terms.

For the research described in this thesis, we have used the CHARMM potential energy function. The CHARMM potential energy function is based on separable internal coordinate terms and pairwise nonbonded interaction terms. It consists of a sum of many terms and is given as follows:

$$E = E_b + E_\theta + E_\varphi + E_\omega + E_{vdW} + E_{el} \tag{2.1}$$

in which the first four terms are the separable internal coordinate terms and the last two the pairwise nonbond interaction terms. The formula for each of these terms, as well as the relevant definitions, are given below.

E_b is the bond potential, which is given by $E_b = \sum k_b(r - r_0)^2$. k_b is a bond force constant dependent on the type of the atoms involved in the bond. The actual bondlength is r, while r_0 is the equilibrium bondlength, i.e., the ideal bondlength for the type of atoms involved in the bond.

E_θ is the bond angle potential, which is given by $E_\theta = \sum k_\theta(\theta - \theta_0)^2$. k_θ is an angle force constant dependent on the type of the atoms that constitute the angle. The actual bondangle is given by θ, while θ_0 is the equilibrium bondangle, i.e., the ideal bondangle for the type of atoms involved in the bondangle.

E_φ is the proper dihedral angle potential, which is given by $E_\varphi = \sum |k_\varphi - k_\varphi cos(n\varphi)|$. k_φ is a dihedral angle force constant, dependent on the type of the atoms that constitute the proper dihedral angle. n is a multiplication factor, which can have the value 2, 3, 4, or 6. φ is the actual proper dihedral angle.

E_ω is the improper dihedral angle potential, which is given by $E_\omega = \sum k_\omega(\omega - \omega_0)^2$. k_ω is an improper dihedral angle force constant, dependent on the types of the atoms that constitute the improper dihedral angle. ω is the actual improper dihedral angle, while ω_0 is the equilibrium improper dihedral angle, i.e., the ideal improper dihedral angle for the types of atoms that constitute the improper dihedral angle.

E_{vdW} is the Van der Waals potential, which is given by $E_{vdW} = \sum_{i \neq j} \left(1/r_{ij}^{12} - 2/r_{ij}^{6}\right)$. r_{ij} is the actual distance between the two atoms for which the Van der Waals potential is calculated.

E_{el} is the electrostatic potential, which is given by $E_{el} = \sum_{i \neq j} q_i q_j / 4\pi\epsilon_0 r_{ij}$. q_i and q_j are the charges of the two atoms for which the electrostatic potential is calculated, r_{ij} is the distance between those two atoms and ϵ_0 is the di-electric constant.

E_b is summed over all pairs of bonded atoms. E_θ is summed over all bondangles. E_φ and E_ω are summed over all proper dihedral angles, respectively all improper dihedral angles. Finally, E_{vdW} and E_{el} are summed over all pairs of atoms. The first four terms force the local structures (bondlength, bondangle, proper dihedral angle, and improper dihedral angle) into their ideal values. The last two terms account for the long range attractive and repulsive interaction forces, that force the local structures out of their ideal values in order to fold the protein in its native state.

2.3.4 The Problem Definition

Now that we have described the potential energy function, we can define the actual problem. Under the assumption that in the native state the potential energy of a protein is globally minimized, we define the problem as follows:

Given the primary structure of a protein, find its tertiary structure using a global optimization algorithm with the given potential energy function as the objective function.

This global optimization algorithm has to be able to solve problems with many more variables than 2-6, since even the smallest proteins have a large number of free variables. Apart from that, the potential energy function is known to have many local minimizers and finding the global minimizer among those is a challenging problem. For proteins, the speculation is that the potential energy function has at least 3^n local minima, n being the number of free variables [LS87] [WCMS88]. Therefore, the protein folding problem belongs to the class of NP-hard problems [CSW94].

The rest of this thesis is concerned with the design and implementation of a global optimization algorithm that is able to solve the protein folding problem.

Chapter 3

Other Approaches

3.1 Introduction

In this chapter we will look at the various approaches that have been developed to solve the protein folding problem. We do not limit ourselves to global optimization methods. Instead, we divide the proposed solutions into three classes: prediction methods, global optimization methods, and alternative methods. Prediction methods are concerned with predicting the native structure of a protein, based on its amino-acid sequence. Global optimization methods are applicable to a wide variety of problems, not just the protein folding problem. The methods in the alternative class are methods not covered by the first two classes. We discuss some of the most important methods from each class in the following sections.

3.2 Prediction Methods

Most prediction methods are concerned with predicting the secondary structure of a protein. They classify each amino acid as part of an α-helix, a β-sheet, or random coil. The purpose of these methods is to identify the regions of secondary structure, in order to be able to build —manually— good initial structures for algorithms that further refine those structures into the native state. In most of the cases, such a refinement algorithm is simply a local minimization algorithm, which is applied to many initial structures. Hopefully, one of the structures after local minimization —specifically, the one with the lowest potential energy— will be the structure of the native state.

Many secondary structure prediction methods have been developed (see [Fas89]). One of the oldest and best known algorithms is the Chou & Fasman algorithm. Recently, neural networks have been used to predict the secondary structure of a protein. Both algorithms are concerned only with the classification of the secondary structure of each amino acid in the protein, and will be discussed briefly below.

3.2.1 The Chou & Fasman Algorithm

The Chou & Fasman algorithm [PF89] is a *statistical* algorithm. Using X-ray crystallograhic technology, Chou and Fasman examined structures of a number of proteins in their native states, and tabulated the number of occurrences of a given amino acid in an α-helix, a β-sheet, and random coil. From this, the conformational parameters for each amino acid were calculated by considering the relative frequency of a given amino acid within all proteins, its occurrence in a given type of secondary structure, and the fraction of amino acids occurring in that type of structure. These conformational parameters are essentially measures of a given amino acid's preference to be found in α-helix, β-sheet, or random coil.

Based on these parameters, Chou and Fasman formulated a set of empirical rules for predicting protein structure (page 394, [PF89]). Using these rules, one is able to predict the secondary

structure of a protein with an accuracy of about 50 percent. The limited accuracy of the predictions is believed to be due to the relatively small number of proteins from which the conformational parameters are calculated. Also, secondary structure might be influenced by the tertiary structure, which is not taken into account at all in this method [HK89].

Despite its limitations, the method is very popular and widely used. Many improvements, which we will not discuss here, have been proposed and implemented. However, the original algorithm still seems to be used the most.

3.2.2 Neural Networks

A neural network is a numerical model of many simple elements, that can be trained from training sets to recognize certain patterns. It consists of a set of computational units, organized in a layered manner. Normally a network consists of an input layer, a hidden layer, and an output layer. Each layer is fully connected to the next layer. There are no connections within a layer. Each connection between a pair of units has a weight attached to it. Each unit has an activation formula, that calculates its output activity from its input activity. The input activity of a unit is the weighted sum of all the output activities of the units from the previous layer connected to this layer. The applied weights are the weights of the connections. Because the weights influence the input activity of a unit —and consequently the activity in the neural network—, we can change the behavior of a network by simply adjusting its weights.

Assigning the desired weights oneself is a nearly impossible task. However, one can *train* a neural network in order to let it find the ideal weights itself. Training is usually done by presenting the network a number of input cases, together with the desired output for each input case. By comparing its output with the desired output, the neural network will measure the error it makes for this particular input. Based on this error it will adjust the weights. Repeating this process will eventually lead to a network that is able to produce the desired behavior. If the training set is a representative subset of the total set, the network is expected to behave correctly for cases that were not included in the training set as well.

Qian and Sejnowski [QS88] were among the first to use a neural network to predict the secondary structure of a protein. Their neural network consists of 17 input units, 40 hidden units, and 3 output units. Each output unit represents one of the possible secondary structures: α-helix, β-sheet, or random coil. During training, they require one of the output units to be 'on' and the others 'off'. The unit that is 'on' indicates in which secondary structure the amino acid resides. Qian and Sejnowski believe the secondary structure of an amino acid is mostly influenced by the 8 amino acids on each side of this amino acid. Therefore, the input for their neural network consists of a sequence of 17 amino acids, the middle amino acid being the amino acid for which the secondary structure has to be determined. To predict the secondary structure for each amino acid in a protein, one has to 'slide' the protein along the neural network.

In order to asses the performance of their neural network, Qian and Sejnowski carefully selected 106 proteins. A subset of these was used to train the network; the rest were used to test the network. This resulted in a performance of about 64 percent correct predictions. Based on these experiments, various other neural networks were tried [HK89] [KCL90] [SLX92], but none of them was able to substantially improve the performance. This seems to be due to the same reasons as the Chou & Fasman algorithm: the limited size of the training set and the influence of tertiary structure on secondary structure. Research is still going on in order to account for these limitations.

3.3 Global Optimization Methods

According to Abagyan, global optimization methods have the largest potential to solve the protein folding problem [Aba93]. Given an empirical energy function, global optimization methods try to find the global minimum of this function. Under the assumption that the empirical energy function models the folded state of a protein exactly, the tertiary structure represented by the global minimum of this function will be the native state of the protein. According to Abagyan

and Argos this can not be assumed [AA92]. Their belief is that the current empirical potential energy functions do not resemble the actual folded state closely enough. Therefore, global optimization methods should create many minimizers, instead of just one (the global minimum). The more general belief though, is that empirical energy functions model the folded state of a protein accurately enough to state that the global minimum of the energy function corresponds to the native state of a protein. In any case, finding many low energy minimizers is considered useful, since this will give more insight in the way proteins fold into their native states.

Many global optimization methods have been developed, but many are only reliable for small-scale problems (i.e., 1-6 variables). However, some methods are applicable to larger problems. The best known are simulated annealing and the genetic algorithm. We will describe both algorithms in greater detail below.

3.3.1 Simulated Annealing

Simulated Annealing, first reported by Kirkpatrick et al [KGV83], is based on a connection between statistical mechanics and the process of crystallization. If a physical system is melted and then cooled slowly, the entire arrangement can be made to produce the most stable (crystalline) arrangement, and not get trapped in a local minimum [WC90]. Given the function to optimize, and initial values for the free variables, simulated annealing starts at a high —artificial— temperature. While cooling the temperature slowly, it repeatedly chooses a subset of the free variables and changes these variables by random numbers. If the objective function has a lower function value, the new variables are chosen to be the initial variables for the next iteration. If the function value is higher, the new variables are chosen to be the initial variables for the next iteration with a certain probability, that depends on the change in value of the objective function and the temperature. The higher the temperature and the lower the change, the more probable the new variables are chosen to be the initial variables for the next iteration. Throughout this process, the temperature becomes lower and lower, until eventually the variables do not change anymore. Then, the function is presumably at its global minimum.

This approach has been used by Wilson and co-workers to find the native state of several proteins [WCMS88] [WC90]. Their results are quite encouraging. They are able to find the native state of most of the smaller proteins (2 to 10 amino acids), using the AMBER potential energy function. However, due to the random nature of the process, multiple runs of the annealing process are usually required to achieve this result. In addition, devising the right cooling schedule is crucial. If the cooling schedule is too fast, the global minimizer of the objective function, and consequently the native state of the protein under consideration, will not be found.

Wilson et al also experimented with their algorithm on larger proteins with 20 to 80 amino acids, but were not as successful as on the smaller proteins. Despite huge amounts of CPU time —sometimes 10 runs of 26 hours each—, their algorithm did not find the global minimizer. Further research is certainly necessary to determine the applicability of simulated annealing to proteins this large.

3.3.2 The Genetic Algorithm

The genetic algorithm is an optimization technique derived from the principles of evolutionary theory. In genetic crossover, the genes of two sources in a population are mixed, thereby creating new genes. These new genes, the children, will have some characteristics of the genes of both parents. In the evolution process, some random mutation of the newly created genes may take place as well. This process of crossover and mutation repeats infinitely in nature. One other important aspect of evolution in nature is the *survival of the fittest principle*. Because of this principle, the genes of the fitter —better— creatures within a population will have more chance to propagate through children than the genes of unhealthy creatures. This implies that through time, the quality of the population increases.

Now, if each member of the population is a distinct set of variables, and if we have an objective function, we can apply this process to the population of variables. We encode the set of variables

of a structure as a gene, with each gene having a corresponding function value (the value of the potential energy function for the values of the variables in the gene). A global optimization algorithm would then consist of repeatedly crossing the genes, randomly mutating the newly obtained genes, and removing some of the genes out of the population that have a bad function value. If we repeat this process long enough, we eventually hope to have a population with genes (i.e., sets of variables) that have a low corresponding function value. The best function value then presumably will be the global minimizer.

Le Grand and Merz applied this technique to the protein folding problem, using the AMBER potential energy function [LM92]. The technique produced the correct results for a protein with 9 amino acids. It also produced very low minimizers for a protein with 5 amino acids, however, these were not obtained consistently. Applied to a protein consisting of 46 amino acids, the algorithm took a month to complete on an SGI IRIS 4D/220, without finding the native state of the protein. Le Grand and Merz suspect that the potential energy function used does not account for all the driving forces of protein folding, explaining why their algorithm did not find the native state of the protein. However, even if the potential energy function would model the driving forces of protein folding correctly, the algorithm still appears to take too much time to be a serious candidate for finding the native state of larger proteins.

3.4 Alternative Methods

Besides the prediction and global optimization methods, research efforts towards solving the protein folding problem over the past years have resulted in a wide variety of approaches [BHW87] [BG85] [PS87] [GLSW92] [HGSA91]. Although some of these methods seem to be promising, researchers have paid more attention to two other types of methods. The first type combines a smart initial protein building phase with an energy minimization phase. The second type uses a lattice based search in order to find the native state of the protein. Both types of methods are described below.

3.4.1 Build-Minimization Methods

Build-minimization methods are among the most popular to solve the protein folding problem. Their objective is to first build a large number of initial protein conformations, each having a different shape. Thereafter, the conformations having the lowest energy value are subject to a local minimization. The hope is that the initial phase creates at least one conformation that has a folded state close enough to the native state, so that the local minimization performed on this conformation will result in the conformation folded in the native state. Of course the success of these methods depends completely on how well the initial protein conformations cover the complete conformational space. Which local minimization technique is used does not significantly influence the results.

Many techniques have been used for building the initial conformations, among which are random creation [Lev83], tree searching [LS88], and dynamic programming [VD90]. However, as soon as larger proteins are addressed, these methods do not perform as well, due to either the randomness of the technique (the first method), or the fact that the search space becomes too large (the latter two methods).

The best known build-minimization method is the one developed by Vásquez and Scheraga [VS85]. They build their initial protein conformations using a simple buildup procedure. Starting with the conformations of individual amino acids, they build dipeptides (a structure consisting of two amino acids) by combining the best individual conformational states. Then, until the conformation of the desired size is formed, they repeatedly build larger conformations by combining the smaller conformations that were formed in the previous iteration. At each iteration, multiple conformations of a certain size are built, but only the best ones (i.e., the ones with the lowest energy value) are eligible for the next iteration. This guarantees that the method will be able to handle larger proteins, since it will not create as many intermediate conformations as other

build-minimization methods. Vásquez and Scheraga used their method to predict the native state of Met-enkephalin, a pentapeptide, in which they succeeded. Despite this result, and the ability of the algorithm to handle larger proteins without requiring huge amounts of CPU time, it is unclear whether the algorithm will perform as well for larger proteins as it does for the relatively small protein Met-enkephalin. This is due to the fact that for larger proteins the conformational space is not completely covered, because only the best intermediate conformations are used to build the larger conformations of the next iteration.

Recent work in the build-minimization area has focused on applying a Monte Carlo chain growth technique [LS87] [BGOV93] [Vel] to build the initial conformations before minimization. The conformation is built atom by atom or amino acid by amino acid, thereby placing the atom or amino acid with a certain probability at a certain position. This probability can be assigned any distribution. Depending on the choice of this distribution, the results of this new method are quite promising, though more research effort definitely is needed.

3.4.2 Lattice Based Search Methods

Lattice based search methods represent the amino acids of a protein as points on a three-dimensional lattice. The lattice usually has the shape of a diamond, although other lattices are in use as well. Each amino acid is encoded to be either H (hydrophobic: lacking affinity for water) or P (hydrophilic: having strong affinity for water), since the belief of the researchers in this area is that these forces completely guide the folding process. The mapping of a protein conformation on the lattice is then a sequence of connected points that are either H or P. Given the lattice, a number of lattice-specific moves are defined. Each move represents a change in the mapping of the conformation on the lattice. Finally, an objective function based on the lattice should be available.

The search on the lattice is usually a Monte Carlo based search. First, a protein conformation is projected randomly on the lattice. Then, for a specific number of iterations, a random amino acid is chosen to be moved. Which move is made depends on a certain probability. Then, after the move has been made, the new energy is calculated. If this energy is lower, the new configuration is accepted as the initial configuration of the next iteration. If the energy is higher, the new configuration is accepted with a certain —low— probability. This process is analogous to the Simulated Annealing process (see 3.3.1).

The lattice based search method was pioneered by Chan and Dill [CD93] and it was refined by Skolnick and Kolinski [SK90] [SK91]. They report some good results. However, as O'Toole and Panagiotopoulos point out [OP92], the results obtained by Skolnick and Kolinski are obtained by biasing the objective function to have certain residues take on certain conformations. This implies that, if the residue happens to be in a different state in the tertiary state of a protein, Skolnick and Kolinski will never find this tertiary state. Without this bias, the results obtained by O'Toole and Panagiotopoulos are not as good. This is due to the fact that by encoding the twenty different amino acids in two key factors (H and P), one loses valuable information that is used by the protein to guide its folding process. If this information could be added to the lattice, a lattice based search method might prove to be a good method to use in solving the protein folding problem.

Chapter 4

Our New Algorithm

In this chapter we describe our newly developed global optimization algorithm. This is done in terms of the protein folding problem, which is the test problem that we used for the algorithm. First we give a brief description of the algorithm designed for the Lennard-Jones, as well as of the algorithm designed for the Water problem, since our new algorithm is based on these algorithms. Then we present a general outline of the new algorithm, hereby noting the differences with both the Lennard-Jones and the Water algorithm. After that, we describe the individual steps of the new algorithm in detail. We describe the alternatives we had to choose from, which heuristics were used, and any further important aspects of the algorithm. In the last section we describe how its parallel implementation influences the algorithm.

4.1 The Lennard-Jones And The Water Algorithms

The basis for our new global optimization algorithm is formed by the algorithms developed for the Lennard-Jones problem [BES92] and the Water problem [BDE+93] [Old93]. These algorithms are reviewed briefly in this section.

The Lennard-Jones and the Water problem are very similar. Both problems have a potential energy function that is *partial separable*. Any function that is the sum of functions, each of which only involves a disjoint subset of the variables, is called partial separable. In the Lennard-Jones and the Water problem, partial separability implies that if a single atom or molecule in a cluster is moved, the potential energy can be re-evaluated cheaply at a cost that is only $2/n$-th of the cost of a total function evaluation, where n is the total number of atoms or molecules in the cluster. This is due to the fact that the potential energy function consists only of sums of pairwise interactions between atoms or molecules. Since not only work is done in the full dimensional parameter space, but also in a small subset of this space, the algorithms developed to solve the Lennard-Jones and the Water problem rely heavily on this property. The subset of the parameter space that is chosen, is problem dependent. In the Lennard-Jones problem, the subset of variables consists of a single atom, whereas in the Water problem, this subset is a single water molecule.

Both algorithms consist of two phases. In terms of global optimization, during the first phase —the sample generation phase— an initial set of minimizers is generated that have a fairly good energy value. In the second phase —the local minimizer improvement phase— special perturbation techniques are used to repeatedly improve local minimizers found in either the first or the second phase.

Since the algorithms to solve the Lennard-Jones and the Water problem are very similar, we will from now on only describe the algorithm to solve the Water problem, hereby where appropriate mentioning the differences with the algorithm to solve the Lennard-Jones problem.

During the first phase of the algorithm, a full dimensional random sample is generated by randomly and independently placing each water molecule. The sample points with the highest potential energies are discarded, and the remaining sample points are improved by using a technique

that randomly samples on, and moves, a single molecule at a time, until the potential energy of the cluster falls below a certain threshold level. From the resulting sampled clusters a subset is selected, and the clusters in this set are used as start points for local minimizations. Some of the local minimizers found in this phase are passed on to the second phase for improvement.

The second phase of the algorithm is executed for some number of iterations. During each iteration the algorithm first selects, using a heuristic, a particular cluster. Thereafter this cluster is expanded relative to the center of mass of the cluster[1]. The molecule that contributes most (or second most) to the potential energy of the selected cluster is chosen, and a global optimization algorithm is applied to the cluster with only this molecule as a variable and the remainder of the cluster fixed. Next, full dimensional local minimizations are applied to the best clusters resulting from the single molecule global optimization step. The clusters with the lowest new potential energies can then be used as start clusters for a following iteration.

Algorithm 4.1 outlines the framework of the global optimization algorithm in terms of the Water problem. Eventually, the best minimizer in the list of full-dimensional local minimizers should be the global minimum. This algorithm has been implemented on a parallel machine, and is exhaustively described in [BDE+93] and [Old93].

4.2 Outline Of The New Algorithm

If we take a look at the protein folding problem, it becomes clear that the Water algorithm as it is can *not* be applied. The main reason is the chain structure of a protein. Moving an atom in the backbone of the protein influences the position of the other atoms along the backbone as well. Therefore, using the internal coordinate system as we do, the potential energy function can not be expressed as a simple sum of pairwise interactions, each of which involves only a small subset of the variables. Hence the partial separability that the Water algorithm utilizes has disappeared, resulting in the need for a new algorithm. 'One-particle sample improvement' (step 1b in the Water algorithm) is not possible anymore. This forces an almost complete new phase 1, because this step was almost solely responsible for the good initial minimizers produced in this phase. Furthermore, expansion is completely impossible: the chain structure of the protein, and the internal coordinate system we use, simply prohibit this step. Finally, the cheap re-evaluation utilized in the 'One molecule global optimization' (step 2c in the Water algorithm) does not exist anymore, implying a new role for this step in the new algorithm.

Before we give an outline of the new algorithm, we have to address some issues concerning the way we use the protein folding problem as a test problem. We noted before (section 2.3.2) that one can choose to represent the protein either in internal coordinates or in cartesian coordinates, and that the number of free variables in both representations is the same. Since it is easier to fix values that are 'natural' to the protein, like bondlengths and bondangles, we apply our algorithm to the protein represented in internal coordinates. In order to reduce the dimensionality of the problem, we fix all internal parameters, except the two proper dihedral angles per amino acid that give the backbone of the protein its shape. All bondangles, all bondlengths, all improper dihedral angles, and the third proper dihedral angle in each amino acid, are fixed at their *ideal* values. This reduces the dimensionality of the problem to about 1/15-th of the original problem, without limiting the folded states a protein can attain.

We can now turn our attention to the new global optimization algorithm, developed to solve the protein folding problem. As in the Water algorithm, the algorithm consists of two phases: a first phase in which initial conformations of the protein are built, and a second phase in which the conformations are improved.

Because it is impossible to move a single atom in a protein chain, the way protein conformations are built in the first phase has changed drastically from the way water clusters are built in the Water algorithm. Instead of randomly sampling each atom, followed by repeatedly executing an improvement step, a conformation is built directly. One by one, each dihedral angle (except the

[1] This was originally not done for the Lennard-Jones problem; research towards using this technique for the Lennard-Jones problem has only recently begun.

Algorithm 4.1

1. **First —sample generation— phase**: Do the following once:
   (a) **Sampling in full domain:** Randomly generate the coordinates of sample clusters, and evaluate the potential energy of each sample cluster. Discard all sample clusters whose potential energies are above a global cutoff level.
   (b) **One-particle sample improvement:** For each remaining sample cluster: while the potential energy is above a global threshold value, and a maximum number of iterations per cluster is not reached, repeat:
      - Select the molecule that contributes most to the potential energy;
      - Randomly sample new locations for the selected molecule;
      - Move the molecule in the sample cluster to the newly sampled location that gives the lowest energy for the cluster.
   (c) **Start point selection:** Select a subset of the best improved sample clusters for local minimizations.
   (d) **Full-dimensional local minimizations:** Perform a local minimization on the coordinates of each start cluster selected. Collect some number of the best of these local minimizers for improvement in phase 2.
2. **Second —local minimizer improvement— phase**: Iterate for some fixed number of times over the following steps:
   (a) **Select a cluster:** From the list of full-dimensional local minimizers, select a local minimizer using a heuristic to determine the most promising candidate.
   (b) **Expansion:** Transform the cluster by multiplying the position of each molecule relative to the center of mass of the cluster by a constant factor (leave the internal structure of each molecule unchanged).
   (c) **One molecule global optimization:** Select the molecule whose partial energy, before expansion, has the worst (or second worst) value. Apply a global optimization algorithm to the expanded cluster with only this molecule as a variable. This global optimization algorithm not only produces the global minimum, but several additional local minimizers as well. Therefore, this step results in several new clusters.
   (d) **Full-dimensional local minimization:** Apply a local minimization procedure, using all molecules as variables, to the lowest clusters that resulted from the one molecule global optimization.
   (e) **Merge the new minimizers:** Merge the clusters with the lowest new potential energies into the existing list of local minimizers.

ones that are fixed) is sampled some number of times, and the sample point that results in the best partial potential energy (calculated from the beginning up to the last atom whose position is determined by the dihedral angle currently worked on) will be chosen. Sampling then continues with the next dihedral angle. Thus, the conformation is built from the beginning to the end without the need of an explicit improvement step. The improvement is an implicit part of the sampling. After a certain number of conformations is constructed this way, some of the best are taken to be subject to a local minimization algorithm. The best resulting conformations —the ones that have the lowest potential energy— will be used as initial minimizers for the second phase.

The second phase of the new algorithm is fairly similar to the second phase of the Water algorithm. As the first step, a local minimizer is chosen to work on. Second, using some heuristic, a small number of dihedral angles is chosen, to which a small scale global optimization algorithm is applied with only these dihedral angles as variables. This global optimization algorithm results in several minimizers, among which the global minimum should be. The best of these minimizers found are then be subject to a full-dimensional local minimization with all dihedral angles as variables. The local minimizers found this way are merged into the list of minimizers. These steps are repeated for some fixed number of iterations. Compared to the Water algorithm, two things have changed notably:

- No expansion is done in the new algorithm. This is impossible due to the fact that a protein has a chain structure.
- Instead of a single molecule, some small number of dihedral angles is chosen to be the free variables in the subsequent small scale global optimization step. This is due to two reasons. First of all, we believe that during the execution of a small scale global optimization, the protein needs to have some flexibility to change its shape. A single dihedral angle does not provide this flexibility. Computational experiments have shown that at least three dihedral angles are necessary. Second, since no cheap re-evaluation of the potential energy function is possible, more use should be made of an expensive step like a small scale global optimization. This is done by letting this part of the algorithm do more work at the same time, i.e., do work on multiple dihedral angles at the same time. The fact that a small scale global optimization has to work on more than one variable is —for small numbers of variables— neglectable.

Despite these changes, the main idea of the second phase —improving existing minimizers by doing some work on a subset of the parameters and some work in the full dimensional parameter space— is still very much there.

Algorithm 4.2 summarizes the above description. As for the Water algorithm, eventually the lowest full-dimensional minimizer in the list of local minimizers should be the global minimum. The corresponding values of the variables then represent the native state of the protein under consideration.

4.3 The New Algorithm In Detail

In this section we will take a closer look at each of the steps of the new algorithm. We will describe the most important aspects of each step, the alternatives we had to choose from, and, if applicable, which heuristics are available.

4.3.1 Step 1a: Create Initial Minimizers

For this step, several methods have been tried, of which the one described as part of algorithm 4.2 gave —as will be shown in chapter 5— the best results. Each of these methods is based on the same idea: build the conformation of the protein dihedral angle by dihedral angle. The methods merely differ in decisions as to whether to continue building the conformation or to back up a dihedral angle in case the partially constructed conformation is bad according to some criterion. The

Algorithm 4.2

1. **First —sample generation— phase**: Do the following once:

   (a) **Create initial minimizers:** Create initial sample points, by building conformations of the protein dihedral angle by dihedral angle. Each dihedral angle is sampled some fixed number of times, and the value that gives the conformation under construction the best partial energy is selected to continue building the conformation.

   (b) **Start point selection:** Select a subset of the best sample points for local minimizations.

   (c) **Full-dimensional local minimizations:** Perform a local minimization on each start point selected. Collect some of the best of these local minimizers for improvement in phase 2.

2. **Second —local minimizer improvement— phase**: Iterate for some fixed number of times over the following steps:

   (a) **Select a conformation:** From the list of full-dimensional local minimizers, select a local minimizer —that was not used before—, using some heuristic to determine the most promising candidate.

   (b) **Select a set of free variables:** Using a heuristic, select some small number of dihedral angles of the protein conformation selected in step 2a to be the free variables in the next step.

   (c) **Small scale global optimization:** Apply a global optimization algorithm to the protein conformation with only the in the previous step selected dihedral angles as variables. The global optimization algorithm not only produces the global minimum, but additional low local minimizers as well. Therefore, this step results in several new protein conformations.

   (d) **Full-dimensional local minimization:** Apply a local minimization procedure, using all dihedral angles as variables, to the protein conformations with the lowest potential energy that resulted from the small scale global optimization.

   (e) **Merge the new minimizers:** Merge the new protein conformations with the lowest potential energy into the existing list of local minimizers.

methods use the notion of partial energy, which is the potential energy of a protein conformation that is built up to a certain dihedral angle. The following methods, each of which will be discussed in full in subsequent paragraphs, have been tried:

1. Sample each subsequent dihedral angle once, until the whole conformation is built, or the partial energy of the part of the conformation built hitherto is above a certain constant threshold. If the partial energy is above the threshold, resample the dihedral angle until the partial energy is below the threshold. If a dihedral angle is resampled some maximum number of times, quit the resampling of this dihedral angle and back up to the previous dihedral angle. Continue the resampling process with that dihedral angle.

2. Sample according to method 1. However, the threshold is not a certain constant threshold, but a threshold that is a combination of a constant threshold, and a value that is based on an initial conformation of the protein. This results in a threshold that follows the pattern of partial energies more closely than a constant threshold.

3. Sample each dihedral angle some number of times, and use the dihedral angle with the lowest partial energy to continue building the protein conformation.

4. Sample each subsequent dihedral angle once, until the whole conformation is built, or the partial energy is above a certain constant threshold. If the partial energy is above the threshold, use the dihedral angle that resulted in the lowest partial energy to continue building the protein. This is in principle method 1, but as soon as the need for resampling arises, we continue sampling as in method 3.

In cases 1 and 2 —where we back up a dihedral angle in case we have sampled a particular dihedral angle the maximum number of times— we start building a new protein if we back up from the first dihedral angle. In each method, we can vary the maximum number of times a dihedral angle can be sampled. Results with varying numbers will be presented in chapter 5.

Construction Method 1

This method was the original method that was to be part of the algorithm. Its idea is simple: if a good value is sampled for a dihedral angle, there is no need for extra work, and we can continue with the next dihedral angle. If a bad value is sampled for a dihedral angle, we simply resample the dihedral angle. If after some maximum number of samples still no good value for the dihedral angle can be sampled, probably one of the dihedral angles earlier in the sample process is the cause, and we back up one dihedral angle to —hopefully— bypass this problem.

For small proteins this strategy works reasonable well. However, for larger proteins, the strategy turns out to be really expensive. It can easily be seen that in the worst case the algorithm is $O(m^n)$, m being the maximum number of samples for each dihedral angle, and n being the number of dihedral angles to be sampled. If the maximum number of samples is 10, and the process has to back up 5 dihedral angles, the number of useless function evaluations is already $100,000$. Backing up 5 steps is certainly not impossible, especially if proteins consisting of for example 40 amino acids, with 80 dihedral angles to be sampled, are built.

Construction Method 2

The second method was ment to be an improvement over the first. Choosing a threshold level in the first method was hard, since during the course of sampling, the further we get with the construction of the protein conformation, the lower the partial energy is[2]. A constant threshold will then be tight at the beginning, but very loose at the end. Using an initial protein conformation,

[2] This is due to the increasing number of long range interactions. These consist of the Van der Waals potential and the electrostatic potential. Both of these usually have negative energy terms, which account for the lower partial energy if the protein is longer.

and offsetting its partial energy at each stage with a constant threshold, should be a better way of thresholding.

However, this method suffers even more from the back up flaw mentioned for method 1. Since the threshold is more tight at the end of the building of the protein, more back up steps will be done, resulting in an even more expensive method.

Construction Method 3

The third method takes a more radical approach. Since the former two methods obviously are not suited for larger proteins, because of their expensive worst case behavior, we needed a cheaper method. Method 3 is such a method, its cost is $O(mn)$, m and n defined as above. A dihedral angle is sampled m times, and the dihedral angle resulting in the lowest partial energy is chosen. Sampling then continues with the next dihedral angle.

This method, as we will see in chapter 5, performs really well, and turns out to be the cheapest method as well. Therefore, it has been made part of algorithm 4.2.

Construction Method 4

Method 4 was an attempt to obtain an even cheaper method than method 3. It is a combination of method 1 and method 3. Method 1 is applied as long as no back up step is necessary, i.e., as long as a good value for a dihedral angle is sampled before the maximum number of samples is reached. If the maximum number of samples is reached, method 4 deviates from method 1. No back up step is done; instead we continue as in method 3.

This method is clearly cheaper than method 3: no extra samples are done in the early stages. However, it turns out that it performs by far not as good as the other methods. Whereas the other methods try to do some improvement in case a bad situation occurs, this method continues with a bad, partially built up conformation. Initial tests led to far inferior sample points; therefore this method was not pursued at all.

4.3.2 Step 1b: Start Point Selection

This step is a rather simple one. All the resulting sample points from the previous step are gathered, and ordered on function value. The k —k can be varied— sample points with the lowest function value will be chosen to be used in the next step.

4.3.3 Step 1c: Full-Dimensional Local Minimizations

In this step, all sample points selected in the previous step are subject to a full-dimensional local minimization. All dihedral angles (except the ones that are fixed) are variables during the local minimization. The resulting minimizers are then collected. Minimizers that are found more than once will be part of the resulting set only once.

4.3.4 Step 2a: Select A Conformation

Before the second phase starts, the minimizers that were collected in step 1c are given a unique label. Thereafter, the minimizers are used to construct the initial list of minimizers. If a minimizer is selected from the list, the new minimizers that stem from it are labeled with the same label as that minimizer and put into the list. However, if a new minimizer is the same as another minimizer on the list, it is discarded. Note, that a minimizer can be selected only once to prevent doing double work.

During the execution of the second phase, for some number of iterations conformations are selected from the list with minimizers according to a balanced selection criterion —the balancing phase—, which is followed by some number of iterations during which conformations are selected according to an unbalanced selection criterion —the non-balancing phase—.

During the balancing phase, care is taken to select the same number of minimizers for each label. This ensures that at least some work is done on each of the minimizers (and its descendants) that were produced during the first phase of the algorithm. During the non-balancing phase, minimizers are selected no matter what the label is. This has the effect that the best minimizers found after the balancing phase are explored in depth.

In the Water algorithm, a fairly complicated selection criterion, based on both the full potential energy and the partial energy of the molecule to be selected in the next step[3], was used. For the protein folding problem we have chosen for a simpler criterion: we simply select the protein conformation that has the lowest potential energy. For the balancing phase this means selecting the conformation with the lowest potential energy among the minimizers with a particular label, for the non-balancing phase this means selecting the conformation with the lowest potential energy among all minimizers.

4.3.5 Step 2b: Select A Set Of Free Variables

This step is one of the most important steps of the whole algorithm. Selecting a good set of free variables to be worked on in the next step, turns out to be crucial to the success of the algorithm. No less than eight different selection methods, each selecting some number m —that can be varied— of dihedral angles, have been tried:

1. Randomly select m dihedral angles;
2. Randomly select a consecutive stretch of m dihedral angles;
3. Randomly select $m/2$ pairs of adjacent dihedral angles (m is restricted to be an even number);
4. Select the m dihedral angles with the worst normalized interaction energy;
5. Select the m dihedral angles using recursive splitting of the search area;
6. Select the consecutive stretch of m dihedral angles that combined have the worst normalized interaction energy;
7. Select the $m/2$ pairs of adjacent dihedral angles that have the worst normalized interaction energies (m is restricted to be an even number);
8. Select the m dihedral angles with the worst interaction energy, using a different normalization strategy.

In the following paragraphs, each of the selection strategies will be briefly explained and its strong and/or weak points will be mentioned.

Selection Method 1: Randomly Select Dihedral Angles

This method was implemented to be a benchmark for the performance of the other methods. It selects the dihedral angles at random, taking no information at all into account. Of course this method is not the best method; however in practice it performs reasonably well. Due to the fact that most of the dihedral angles are not at their optimal value in the early stage of the improvement process, the method performs well in this stage. At a later stage however, this selection method drops in performance drastically. This is due to the fact that in the later stage specific dihedral angles need to be worked on. The chance to select only those bad dihedral angles becomes lower and lower.

[3] This is possible due to an implementation 'trick', that will not be further discussed here.

Selection Method 2: Randomly Select A Consecutive Stretch Of Dihedral Angles

Randomly selecting a consecutive stretch of dihedral angles was ment to be an improvement over method 1. The restriction of selecting a stretch should give a conformation more possibilities to fold itself in the next small scale global optimization step. However, this method suffers from the randomness of the selection as much as method 1, and turns out to produce comparable results.

Selection Method 3: Randomly Select Pairs Of Adjacent Dihedral Angles

Randomly selecting pairs of adjacent dihedral angles was ment to be an improvement over method 1 as well. Selecting dihedral angles of pairs should —again— give a conformation more possibilities to fold itself in the next small scale global optimization step. However, like methods 1 and 2, this method suffers from the randomness of the selection, and does not give any improvement in overall results of the algorithm compared to selection method 1.

Selection Method 4: Select The Dihedral Angles With The Worst Normalized Interaction Energy

This method requires some more explanation than the former three. The basis of this method is formed by the notion of *interaction energy*. Given a dihedral angle, this is the energy between the atoms on the left of the dihedral angle and the atoms on the right of the dihedral angle. If we normalize the interaction energy of each dihedral angle by the product of the number of atoms on the left and the number of atoms on the right, we are able to make a fair comparison between the normalized interaction energies. This is exactly what is done in method 4: the dihedral angles that have the highest normalized interaction energy are selected.

Selecting the dihedral angles this way turns out to be very effective: this method consistently produces better results than the other seven methods. It seems to be able to select the right dihedral angles at the right moment, thereby allowing the next small scale global optimization step to improve the folded state of a conformation considerably. Characteristic is that the method has a tendency to select dihedral angles in small groups. This seems to be what is needed to let a conformation fold itself properly during the course of the algorithm.

Selection Method 5: Select The Dihedral Angles Using Recursive Splitting Of The Search Area

Despite the fact that it seems that grouping of dihedral angles seems to work very well, we wanted to make an experiment in which we can guarantee that the dihedral angles that are selected will be distributed over the whole conformation. Method 5 provides a way of doing this by using a recursive splitting technique. Given the normalized interaction energy of each dihedral angle, the dihedral angle with the worst normalized interaction energy is selected. Then, half of the remaining number of dihedrals to be selected, will have to be selected on the left of this dihedral angle, and the rest on the right of this dihedral angle. The selection of the remaining dihedral angles in a particular region is done the same way: the dihedral angle with the worst normalized interaction energy is selected, and the remaining number of dihedral angles to be selected in that region is distributed evenly over the two new regions. This process is repeated until eventually no dihedral angles are left to be selected.

The results obtained using this method are better than the results obtained using method 1. However, they are not as good as those obtained by method 4. This seems to be due to the fact that some of the dihedral angles selected are indeed 'bad' dihedral angles, in a sense that they need to be selected in order to change the folded state of a conformation significantly. However, some of the dihedral angles selected will not contribute to this process. This is due to the fact that they are selected in a region where no 'bad' dihedral angles are. It seems that distributing the dihedral angles over a conformation is not as good as the grouping observed in method 4.

Selection Method 6: Select The Consecutive Stretch In Which The Dihedral Angles Combined Have The Worst Normalized Interaction Energy

Selecting a consecutive stretch in which the dihedral angles combined have the worst normalized interaction energy is —in respect to method 4— exactly the opposite of method 5. In this method the observed subgrouping of method 4 is taken into extreme. The normalized interaction energies of each consecutive group of some number of dihedral angles is summed up, and the dihedral angles from the group with the highest normalized interaction energy are selected. Note that the various groups overlap each other.

Tests with this method produced results comparable to method 5. Due to the same reason —some 'bad' and some 'good' dihedral angles are selected—, the results are not as good as those obtained by applying method 4.

Selection Method 7: Select Pairs Of Adjacent Dihedral Angles That Combined Have The Worst Normalized Interaction Energy

The worst pairs of adjacent dihedral angles are selected in this method. Based on the same argument as used for method 3, this method resembles the grouping observed in method 4 very close. For each pair of adjacent dihedral angles the combined normalized interaction energy of the two dihedral angles is calculated. Thereafter, some number of the worst pairs are selected to be worked on in the next small scale global optimization step.

Again, this method does not produce results as good as method 4 does. Although it results in lower minimizers than methods 5 and 6, the values found are still higher than the ones found with method 4. The same argument as for methods 5 and 6 can be applied. However, since pairs are selected, less already 'good' —for which no improvement is needed— dihedral angles will be chosen. Hence the improvement over method 5 and method 6.

Selection Method 8: Select The Dihedral Angles That Have The Worst Normalized Interaction Energy, Using A Different Normalization Strategy

Method 8 resembles method 4 very closely. The only difference is the normalization strategy. The interaction energy is not normalized by the product of the number of atoms on the left and the number of atoms on the right of the dihedral angle. Instead, it is normalized by the maximum of those two numbers. This strategy is designed to examine the behavior of a different normalization.

Using this method all by itself does not produce very good results. Since most of the selected dihedral angles are at the ends of a protein conformation, the middle part never gets optimized, resulting in conformations that are by far not completely folded in their native state.

However, this method is suited for finally polishing both ends of a conformation. Selection method 4 has a slight bias towards selecting in the middle of a conformation, therefore method 8 is perfect to polish the minimizers that were optimized using method 4.

4.3.6 Step 2c: Small Scale Global Optimization

The protein conformation selected is subject to a small-scale global optimization with the selected variables as variables. Since the number of variables selected in step 2b is generally small, the global optimization technique used is reliable and bound to find the global minimum for this problem. In addition, the global optimization technique yields also several additional low local minimizers. This global optimization technique is described exhaustively in the work of Smith et al. [SES89] [SS92]. Many parameters can be set for this algorithm, but since it is only a small[4] part of our algorithm, we will not discuss these here. It suffices to say that we use the static, asynchronous version, and that the parameter setting used is reliable in a sense that presumably the global minimizer is among the several minimizers that result from executing the algorithm.

[4] Though important!

4.3.7 Step 2d: Full Dimensional Local Minimization

Since a global minimum in the small parameter space in general does not correspond to a local minimizer in the full parameter space, the best small-scale minimizers resulting from the previous step are subjected to a local minimization. As in step 1c, all dihedral angles are variables in this step, giving the protein a change to further polish itself in response to the new values of the dihedral angles that were optimized in the previous step. The same local minimization algorithm is used as in step 1c, so no further discussion is needed here.

4.3.8 Step 2e: Merge The New Minimizers

In this final step of phase 2, the full-dimensional local minimizers resulting from step 2d are merged into the list of minimizers found so far.

4.4 The New Algorithm In Parallel

Algorithm 4.2 as stated is a sequential algorithm; however, it has been implemented as a parallel program. This has several effects on the algorithm: in general the parallel version behaves differently than the sequential version, and will generate different results. Of course the global minimum should be among the minimizers produced, but the path of intermediate minimizers through which it is reached can be different. In this section we address these issues. First we describe the parallelism in the first phase of the algorithm, after which we describe the parallelism in the second phase. We do not describe parallelism in general; the reader is referred to the wealth of literature that is available on this topic (see for example [GL88] [LER92] [AG94]). In the next sections we discuss differences between the sequential execution and the parallel execution. It should be noted that since the algorithm is based on random sampling, two different runs can produce different results. Therefore, if we talk about differences between algorithms, we abstract from this fact. We will treat the sequential and parallel versions of the algorithm as producing the same, deterministic results.

4.4.1 Parallelizing The First Phase

Parallelizing the first phase is a rather simple task. We have created a two-layer structure. The first layer consists of a single central scheduler. The second layer is a collection of worker processes. Suppose we have n worker processes, m sample points to create ($m \geq n$), and k of those sample points have to be used as start points for a local minimization ($n \leq k \leq m$). Each worker creates m/n sample points. Thereafter, each worker selects k/n of the sample points it generated, and sends them to the central scheduler. The central scheduler collects the sample points, and as soon as a worker is idle, it hands this process a sample point to do a local minimization on. After a worker finishes a local minimization, it sends the resulting local minimizer to the central scheduler. The central scheduler keeps on collecting local minimizers and handing out sample points to idle workers, until eventually no more tasks (i.e., no more sample points to perform a local minimization on) are to be done. The list of the collected minimizers is then stored in such a way that the second phase can use the results from the first phase.

The parallel version of the algorithm will produce different results than its sequential counterpart, simply because the algorithm is slightly different. In the sequential algorithm, all sample points are first collected, and then the k best sample points are selected for a further local minimization. The parallel algorithm instead selects the k/n best sample points at each worker. It should be clear that the sample points selected can very well be different. For example, if the third best minimizer at worker 1 is better than the best minimizer at worker 2, this happens. Therefore, the parallel version of the algorithm is not guaranteed to select the k best minimizers.

Except for this issue, there is the issue of the algorithm being synchronous or asynchronous. If the algorithm would have been synchronous, we could have easily sent all sample points to the central scheduler, who would then pick the k best. This is the same as the sequential algorithm.

However, we have implemented our algorithm as an asynchronous algorithm. The advantage of this scheme is that —except for when there is no task to be done— no process will be idle during the execution of the algorithm. It is possible for a certain worker to eventually do local minimizations on sample points generated by another worker. Especially if the process of generating sample points is not deterministic (as in the case of for example construction method 1, as described in section 4.3.1). A possible scenario is that all workers but one are done, and that this worker is still generating sample points. Once this worker is done, its k/n sample points that are subject to a local minimization will be handed out to as many workers as needed. This guarantees a faster termination of the whole algorithm, than if the generating worker would have done all local minimizations himself.

4.4.2 Parallelizing The Second Phase

The parallel version of the second phase is rather complicated, because of its three-layer structure. The first level is the central scheduler, that takes care of phase 2, parts a, b, and e (selecting a conformation, selecting a set of free variables[5], and merging the new minimizers). Once a conformation and a set of free variables are selected, it hands this conformation and the set of free variables over to a second level process —hereafter to be called a foreman—, takes care of phase 2, parts c and d. Each second level process works in cooperation with some number of processes at the third level, the so-called worker processes. Each group, consisting of one foreman and some workers, first does the small scale global optimization process in cooperation. We will not go into the details of this process. For an excellent description the reader is referred to [SES89] and [SS92]. After the small scale global optimization, each process in the group, including the foreman, keeps on doing full-dimensional local minimizations until no more small scale global minimizers need to be polished. Each worker then sends the results of the full-dimensional local minimization to the foreman. The foreman merges the received local minimizers with the minimizers he found himself, and sends all the minimizers back to the central scheduler.

In this manner, many subgroups consisting of a foreman and some workers are improving different minimizers at the same time. Within each group, the execution of the task at hand is synchronous. Only after the whole group is done with all its local minimizations, the local minimizers are sent back to the central scheduler, and the group will get a new minimizer to improve. Avoiding the idle time that is created this way is a very difficult task, and would result in a completely different algorithm.

At the higher level the algorithm is asynchronous again: the central scheduler does not wait until every group has completed its task before handing out new tasks. Instead, as soon as a group is done, the group will get a new task to work on.

Two closely related issues arise. First of all, many minimizers are worked on at the same time. Second, the algorithm is asynchronous. Already the fact that many minimizers are worked on at the same time accounts for a different execution of the algorithm than the sequential version. Instead of working on the one best minimizer, the parallel version works on many good minimizers at the same time. It should be clear, that if one of the groups produces a better minimizer than a minimizer that was handed to another group to work on, the parallel version behaves different. In the sequential case, the minimizer that was originally handed to the other group would not have been selected at this particular point in the computation, in favor of the new, better minimizer.

The fact that the algorithm is asynchronous adds to this effect. It is now very much possible that at one point in time a minimizer is handed out to some group, and that right after that new minimizers arrive at the central scheduler that have far better values. Clearly this is a disadvantage, and it votes for a synchronous version of the algorithm. There, at each synchronization point, the best minimizers would be selected. However, in the synchronous case of the algorithm, a lot of processing time would be wasted. Each subgroup, working on a different minimizer, probably needs a different execution time. Therefore, we have opted for the implementation of an asynchronous second phase of the algorithm.

[5] This can be done in a lower level process as well. However, the implementation as it is right now has this step as a task for the central scheduler, due to some other experiments that are not discussed here.

Chapter 5

Computational Results

5.1 Implementation

Our new algorithm has been implemented on a KSR (Kendall Square Research) parallel supercomputer [Cor92] [Cor91], that has 64 processors. The individual processors are organized in two —connected— rings of 32 processors each. Each processor has 32 megabytes of local memory. However, the memory in the KSR is treated as being global: every processor has access to all memory. The distribution of data among the memory is taken care of by the machine, the user has no influence on this.

Our algorithm has not been designed with this specific parallel machine in mind. Instead, we have tried not to be bound to a particular machine at all. Therefore, we have chosen to let processes communicate through the P4 communications library (version 1.3 [P4]). This is a library that is portable across many different parallel (super)computers. Our implementation of the new algorithm should therefore be easy to adapt to new computers.

Some parts of the algorithm are implemented in the programming language FORTRAN. However, the main part of the algorithm is written in the programming language C. The code to evaluate the potential energy of a protein was supplied to us by Teresa Head-Gordon, and is the potential energy evaluation routine of the CHARMM package [BBO+83]. To evaluate the potential energy of a protein, this package requires an input file with all the necessary parameters —that can be modified according to ones personal taste— for the CHARMM potential energy function[1].

5.2 Results Of The First Phase

In this section we present the results obtained by executing the first phase of our algorithm. We first compare the results of two of the heuristics discussed in section 4.3.1. Thereafter we present results obtained for various size proteins. Those results will be used in the subsequent second phase of the algorithm.

5.2.1 Comparison Of The Heuristics In The First Phase

In this section we compare the results of the following two heuristics (each described in section 4.3.1):

1. Sample each subsequent dihedral angle once, until the whole conformation is built, or the partial energy of the part of the conformation built hitherto is above a certain constant threshold. If the partial energy is above the threshold, resample the dihedral angle until the partial energy is below the threshold. If a dihedral angle is resampled some maximum

[1] The input files we used for the various size proteins can be obtained by contacting the author.

threshold value	*max resamples*	*sample value*			*minimizer value*		
		min	*max*	*avg*	*min*	*max*	*avg*
25.0	5	-111.05	-18.43	-51.74	-360.38	-210.74	-291.26
	10	-85.97	-9.74	-48.00	-373.47	-214.94	-296.42
	20	-104.02	-17.47	-40.94	-358.07	-240.88	-304.52
100.0	5	-25.11	55.65	20.18	-336.49	-194.46	-280.17
	10	-3.96	55.74	26.43	-339.04	-184.01	-281.87
	20	-46.02	69.00	21.51	-352.37	-198.75	-275.22
1000.0	5	141.06	874.99	449.91	-343.47	33.76	-185.59
	10	25.71	709.09	449.04	-352.56	-40.79	-191.11
	20	-27.28	690.65	425.22	-344.77	89.68	-208.61

Table 5.1: Results of sampling including the back up step. Listed are the minimum, maximum, and average values of the sample points and of the local minimizers resulting from a subset of these sampled points.

number of times, quit the resampling of this dihedral angle and back up to the previous dihedral angle. Continue the resampling process with that dihedral angle;

2. Sample each dihedral angle some number of times, and use the dihedral angle with the lowest partial energy to continue building the protein conformation.

Preliminary testing of the other two heuristics mentioned in section 4.3.1 indicated that their results were inferior to the two heuristics listed above. We explained the reasons for this inferior behavior in section 4.3.1, and do not discuss the two alternative heuristics any further here.

To compare the results of both heuristics, we applied the first phase of the algorithm to poly(20-Ala), a protein consisting of 20 amino-acids of the type Alanine. This protein is known to have an α-helical tertiary structure, and its size should be small enough for the first phase of the algorithm to find this tertiary structure. We used 20 worker processes for each run. Each worker sampled 10 conformations, and sent the best 2 conformations to the central scheduler. The 40 conformations were then used as start points in the subsequent local minimizations.

Both heuristics have been tested with various parameter settings. Heuristic 1, that includes the back up step, has been tried with various thresholds and various numbers for the maximum number of resamples. Heuristic 2 has been tried with various numbers for the number of times a dihedral angle is sampled. We will present the results of each heuristic, after which we will make a comparison between the results obtained from each heuristic.

Results Sampling Each Dihedral Angle Including A Back Up Step

Heuristic 1, sampling each dihedral angle including a back up step, has been tested with the following three constant thresholds: 25.0, 100.0, and 1000.0[2]. Furthermore, three maximum numbers for the number of resamples have been applied: 5, 10, and 20. In table 5.1 we list the results obtained with the various settings of these parameters. For each combination of the threshold level and the maximum number of resamples, we list some key values for the sample points that were created (a total of 200), and some key values for the local minimizers that resulted from applying a local minimization algorithm to the best sample points (a total of 40). The key values we list are the minimum —best— value found, the maximum —worst— value found, and the average value found.

Looking at table 5.1 we note some interesting results. First of all, it is definitely worthwhile from the prospective of the average minimizer values found to use a low threshold value, i.e., a threshold value of 25.0. It is clear that the average minimizer values produced by working with a higher threshold are not as good as the ones found with a threshold value of 25.0. A threshold value of 100.0, combined with a maximum number of resamples of 20, produces an average minimizer

[2] All energy values we present in this chapter are in kcal/mole.

no of samples	*sample value*			*minimizer value*		
	min	*max*	*avg*	*min*	*max*	*avg*
3	-257.83	99.78	-132.93	-381.16	-225.16	-327.59
5	-288.63	-108.26	-237.20	-391.27	-317.57	-349.80
10	-341.55	-255.36	-297.68	-405.11	-338.11	-363.56
20	-431.99	-319.77	-366.68	-476.09	-348.94	-418.53

Table 5.2: Results of sampling each dihedral angle some number of times. Listed are the minimum, maximum, and average values of the sample points and of the local minimizers resulting from a subset of these sampled points.

value that is 29.30 ($-275.22 - -304.52$) higher than the average minimizer found with a threshold value of 25.0 with the same maximum number of resamples.

Furthermore, increasing the maximum number of resamples does *not* seem to influence the sampled values found by this heuristic all that much. If we look at the average sampled value with a threshold of 25.0, we see that the average sampled value goes up as the maximum number of samples is increased. However, with a threshold of 1000.0 the average sampled value goes down as the maximum number of samples is increased. Also, if we look at the average value of the minimizers found, we see that in the case of a threshold of 25.0 the average value goes down, and in the case of a threshold of 100.0 the average value has a random behavior. It seems the differences in these values —sample values and minimizers—, are due to the randomness of the sampling. The maximum number of resamples —at the values chosen here— does not seem to influence these values.

If we set the maximum number of resamples to 0, the chance of sampling a conformation becomes lower and lower as the protein becomes larger and larger. Therefore, the number of maximum resamples should be at least higher than 0. However, as we saw above, above a certain —unknown[3]— level, the number of resamples does not matter. This can be explained as follows: while building a protein, it is nearly impossible to continue building the conformation if a bad value for a dihedral angle is sampled. Every sampled value for the next dihedral angle will probably result in a partial energy that is higher than the threshold level. Increasing the maximum number of resamples does not influence this. However, the maximum number of resamples still should be above 0, to give the algorithm a chance to sample a reasonable value for each dihedral angle.

Results Sampling Each Dihedral Angle Some Number Of Times

Heuristic 2, sampling each dihedral angle some number of times and selecting the best one, has been tested with the following number of times a dihedral angle should be sampled: 3, 5, 10, and 20. In table 5.2 we list the results obtained with the various settings of this parameter. For each number of samples we list some key values for the sample points that were created (a total of 200), and some key values for the local minimizers that resulted from applying a local minimization algorithm to the best sample points (a total of 40). The key values we list are the minimum —best— value found, the maximum —worst— value found, and the average value found.

The results presented in table 5.2 are predictable. Increasing the sample density for each dihedral angle from 3 to 20 results in an improvement in the values sampled. In fact, the improvement is considerable. If we look at the best sample value found with 20 samples per dihedral angle (-431.99), we see that the value is very close to the best value found by the local minimizations (-476.09). It turns out, as we shall see in the section that presents the results of phase two of the algorithm (in section 5.3.2), that the value -476.09 is the global minimum for this particular problem. Sampling using the second heuristic therefore proves to be a very effective method.

[3] We were not interested in determining this barrier.

heuristic	*elapsed time (sec)*			*no of func evals*		
	min	*max*	*avg*	*min*	*max*	*avg*
1	16	2410	266	143	18708	2245
2	78	134	80	721	721	721

Table 5.3: The cost of the sampling in phase 1 of the algorithm for each heuristic.

heuristic	*elapsed time (sec)*			*no of func evals*		
	min	*max*	*avg*	*min*	*max*	*avg*
1	784	3582	1691	2546	11594	5447
2	500	2093	1040	1641	6868	3409

Table 5.4: The cost of the local minimizations in phase 1 of the algorithm on the results of sampling with each heuristic.

Comparison Of The Results Of Both Sample Heuristics

If we compare the results of the two heuristics, it is clear that the results of the second heuristic —sampling each dihedral angle some number of times and selecting the best value to continue building the protein— are superior to the first heuristic. However, this does not necessarily imply that the second heuristic should be included in the algorithm. To be included, the second algorithm should also have at least a *cost* comparable to the first heuristic. Table 5.3 presents various measurements that were done to identify the cost of sampling using each heuristic. Table 5.4 presents various measurements of the local minimization step, that was executed on the sample points that resulted from using each heuristic. In both tables we compare the 20 resample, 25.0 threshold version of heuristic 1 with the 20 sample version of heuristic 2.

If we look at table 5.3 and 5.4, we can see that, for both the sampling part and the local minimization part of phase 1, the average cost of the second heuristic is lower than the cost of the first heuristic. Furthermore, we also note that the cost of the second heuristic is more balanced than the average cost of the first heuristic. This is due to the fact that the number of functions evaluations in the second heuristic is the same for every sample point generated. For the first heuristic, this is not true: backing up within this sample process makes that the number of function evaluations can vary at great length.

The local minimization statistics are more similar, although the local minimizations on the sample points generated by the second heuristic are almost twice as cheap (i.e., slightly more than half the number of function evaluations). This seems to be due to the fact that the resulting sample points from the second heuristic are better than the resulting sample points of the first heuristic. Once more, this indicates that the second heuristic is better than the first.

5.2.2 Results Of The First Phase For Various Size Proteins

We have applied the first phase of our algorithm to several proteins of the type poly(L-Ala), each having a different size. Table 5.5 lists the results obtained for each size protein. Again we differentiate between the sampled values and the values resulting from the subsequent local minimizations. We list the minimum, maximum, and average value found for both types of values. For the sizes 3 till 20, the number of sample points was 200, and the number of local minimizations 40. For the sizes 30 till 58, the number of sample points was 250, and the number of local minimizations 50. In both cases, 1 central scheduler was active. In the first case, the number of worker processes was 20, in the latter we used 25 worker processes.

From table 5.5 it is clear that the larger the protein, the more effect the local minimization has on the sampled values. For example, for the protein of size 58, the average value after local minimization is 301.99 better than the average value after sampling. Compared to a difference of only 1.58 in the case of a protein of size 3, this is a big difference. Relatively speaking, the average value for proteins of size 58 decreases with approximately 33 percent, while for the proteins of

protein size	*sample value*			*minimizer value*			*average improvement*
	min	*max*	*avg*	*min*	*max*	*avg*	
3	-28.10	-24.43	-26.57	-28.19	-26.75	-28.15	1.58
5	-69.79	-58.08	-63.91	-76.17	-63.24	-70.80	6.89
10	-186.67	-145.52	-162.67	-195.09	-157.77	-180.40	17.73
20	-412.51	-325.47	-365.82	-476.09	-338.11	-418.53	51.78
30	-669.03	-466.74	-551.53	-748.70	-557.53	-646.33	94.80
40	-879.93	-396.56	-717.69	-998.13	-736.95	-863.85	146.16
58	-1233.77	-253.46	-982.46	-1412.23	-1121.13	-1284.45	301.99

Table 5.5: Results of the first phase of our algorithm for various size proteins of type poly(L-Ala).

protein size	*sampling*		*local minimization*	
	avg no of func evals	*avg time (sec)*	*avg no of func evals*	*avg time (sec)*
3	41	0	19	0
5	121	0	31	1
10	321	6	66	8
20	721	53	126	54
30	1121	190	164	150
40	1521	458	235	383
58	2241	1422	374	1286

Table 5.6: Cost the first phase of our algorithm for various size proteins of type poly(L-Ala).

size 3 the average value decreases about 5 percent. This is due to the fact that for small proteins the sampled conformations are much better than for large proteins: for larger proteins it is much harder to sample a good value for each dihedral angle, while for a small protein this is more likely due to the low number of dihedral angles. Therefore, the local minimization can improve on the larger proteins relatively more than on the smaller proteins.

In table 5.6 we list the cost of both the sampling part and of the local minimization part of phase 1. For each part we list the average number of function evaluations and the average time (in seconds). It is clear from the first two columns, that the function evaluation is of $O(n^2)$. However, if we look at the latter two columns, the average time needed for a local minimization increases more than the average time needed for sampling. This is due to the fact that the local minimization needs to perform some linear algebra calculations at each iteration of the local minimization, that are of $O(n^2)$ as well. This effects the average time per local minimization considerably.

5.3 Results Of The Second Phase

In this section we present the results obtained by executing the second phase of our algorithm. We first compare some of the heuristics to select a set of free variables that were presented in section 4.3.5. Thereafter, we present the results of applying the second phase of the algorithm to the minimizers found in the first phase of the algorithm.

5.3.1 Comparison Of The Heuristics In The Second Phase

In this section we compare some of the heuristics that were presented in section 4.3.5. To this extend we designed an experiment in which we let the first phase of our algorithm produce real high minimizers. We only sampled each dihedral angle once, and performed a single local minimization on each of the sample points. The local minimizers that resulted were used as a test case for the various heuristics to select some number of dihedral angles in the second phase of the algorithm.

We used poly(20-Ala) as a test case, and let our second phase run on 61 processes: 1 central

heuristic	*best minimizer*
1	-415.17
2	-458.03
3	-425.65
4	-418.09
5	-454.18

Table 5.7: Results of the second phase for various heuristics to select dihedral angles.

scheduler, 15 foremen with each 3 workers. A total of 250 iterations, 125 balanced and 125 non-balanced, were executed. The following heuristics (see section 4.3.5 were compared:

1. Randomly select m dihedral angles;
2. Select the m dihedral angles with the worst normalized interaction energy;
3. Select the m dihedral angles using recursive splitting of the search area;
4. Select the consecutive stretch of dihedral angles that combined have the worst normalized interaction energy;
5. Select the $m/2$ pairs of adjacent dihedral angles that have the worst normalized interaction energies.

The other three heuristics mentioned in section 4.3.5 were not examined. The two random heuristics because initial tests proved that they produced similar results as the random heuristic mentioned here; the heuristic with the alternative normalization strategy because initial tests proved that the heuristic produced worse results than the random heuristic mentioned here.

In comparing the heuristics, our criterion is the best minimizer produced by the various heuristics. In table 5.7 we list the best minimizers found during a typical run of phase 2 with each heuristic. We have run phase 2 several times with each heuristic. The values presented here are taken from one run, and represent the results of the other runs accurately.

The results prove what has been stated in section 4.3.5. Selecting a consecutive stretch of dihedral angles (heuristic 4), or dihedral angles that are spread out too much (heuristic 3), is not favorable. Though favorable over the random selection (heuristic 1), it is not as good as the selection of worst pairs of adjacent dihedral angles (heuristic 5), or the selection of individual worst dihedral angles (heuristic 2). Selecting pairs of adjacent dihedral angles that have the worst normalized interaction energy is not as good as selecting individual dihedral angles that have the worst interaction energy. If we analyze the dihedral angles chosen by the latter one, it turns out that it chooses dihedral angles in small groups. Selecting consecutive pairs mimics that behavior, but lacks the flexibility to select for example a group of 3 dihedral angles. Hence the results for this heuristic are slightly worse.

5.3.2 Results Of The Second Phase For Various Size Proteins

We have applied the second phase of our algorithm to the minimizers found in the first phase. We used the selection method that selects the dihedral angles that have the worst normalized interaction energy to select 5 dihedral angles at each iteration. A total of 250 iterations, 125 balanced and 125 non-balanced, were executed by 61 processes: 1 central scheduler, and 15 foremen that each had 3 workers.

Table 5.8 presents the best minimizer found in the first phase, and the best minimizer found in the second phase after 250 iterations executed as described above. Table 5.10 presents the cost of executing phase 2 for the various size proteins[4].

[4] Unfortunately, the data for the cost of poly(30-Ala) were lost.

protein size	*best minimizer phase 1*	*best minimizer phase 2*
3	-28.19	-28.19
5	-76.17	-76.17
10	-195.09	-195.09
20	-476.09	-476.09
30	-748.70	-748.70
40	-998.13	-1045.41
58	-1412.23	-1500.93

Table 5.8: The best minimizers found in the first phase of the algorithm, and after 250 iterations of the second phase of the algorithm.

First we note from the results listed in table 5.8, that up to a protein size of 30, the second phase does not improve over the first phase. For the proteins of size 3, 5, 10, and 20, this is because the first phase already finds the global minimizer for this problem. The values of the free dihedral angles of the proteins of size 3 and 5 are equal to the ones listed in [HGSA91]. The best minimizer we found for the protein of size 10 is in a perfect α-helical shape, which is in accordance with the results presented in [WC90] and the predictions made in [Vel] and [BGOV93]. For the protein of size 20, our results are in agreement with [WC90] and [BGOV93].

Despite the fact that the second phase does not improve over the first phase, the protein of size 30 is not in its global minimum yet. If we examine the structure found, it turns out that the ends of the conformation are not in their most favorable position. This is due to the fact that the selection strategy used has a small bias towards selecting dihedral angles in the middle of the protein instead of at the ends. If we apply one or two more iterations of the second phase with a different selection strategy —the one that selects the dihedral angles with the worst interaction energy based on a different normalization—, we easily find the global minimum for this problem. The energy in the global minimum is then -760.49. The conformation is in a perfect α-helical shape.

In the case of the protein of size 40 the second phase of our algorithm shows, for the first time, an improvement over the first phase. The resulting minimizer is a perfect α-helix. The ends of the conformation are in a correct shape as well, no extra work needs to be done there. [WC90] reports that they found most of their best conformation to be in α-helical shape, which is a clear indication that the whole protein in the native state should be in this shape. Their algorithm was not able to find the native state; however, our algorithm shows that the native state of poly(40-Ala) is purely α-helical.

Poly(58-Ala) is the most interesting case. Weili and Cui [WC90], using their simulated annealing algorithm, find α-helical structures for poly(L-Ala) up to 80 amino-acids. However, Head-Gordon and Stillinger [HGS92] show that for poly(58-Ala) a lower minimizer exists: two parallel α-helices, connected by a β-bend. The best minimizer we have found, after 250 iterations of the second phase of our algorithm, has this particular shape. However, it is by far not the global minimizer, since the α-helices are not completely formed yet. A continuing experiment, with 125 extra non-balanced iterations on the best minimizers found after 250 iterations, resulted in a best minimizer that had a function value of -1523.24 and that was shaped like a single, straight, not completely formed, α-helix.

Not satisfied with this result, we did another experiment, in which we started again with the best minimizers after 250 iterations of the second phase. We ran again 125 extra non-balancing iterations. The resulting best minimizer from this experiment was different than the one resulting from the previous, exactly the same, experiment. The best minimizer resulting from this experiment had a function value of -1523.09. The protein corresponding to this function value, has again two parallel α-helices that are connected by a β-bend. However, this minimizer is not completely formed yet either, since the α-helices at the end are not in α-helical shape yet.

Clearly, the global minimizer for poly(58-Ala) has not been found yet by our algorithm. Ho-

protein size	*best minimizer*	*global minimizer*
3	-28.19	yes
5	-76.17	yes
10	-195.09	yes
20	-476.09	yes
30	-760.49	yes
40	-1045.41	yes
58	-1523.24	no

Table 5.9: The best minimizers found by our algorithm. We also list whether we believe the best minimizer found so far is also the global minimizer.

protein size	*no of func evals*	*no of iterations*	*avg time per iteration (sec)*
3	48448	4	74
5	326878	38	134
10	541526	54	575
20	2695783	250	1169
30	–	–	–
40	2769537	250	3900
58	2937872	250	9263

Table 5.10: Cost of the second phase of our algorithm for various size proteins of type poly(L-Ala).

wever, we think the basic shape of the global minimizer has been found, and we expect that continuing experiments[5] will find the global minimizer for this protein.

A summary of the best minimizers found so far for the various size proteins is given in table 5.9. We also list whether we believe whether the best minimizer listed is also the global minimizer. For the poly(L-Ala) proteins of size 3 to 40 we think that we found the global minimizer. For poly(58-Ala) we stated before that we need do to more work.

If we look at table 5.10, we first note that for the proteins of size 3 to 10, less iterations than 250 are executed. This is because the algorithm can not find any new minimizers, that weren't already in the list. Eventually, no minimizer, that has been worked on before, can be selected. The algorithm then stops executing. Therefore, the number of iterations is less than 250.

Furthermore, we note from table 5.10 that the cost of the second phase is considerable. For example, for the protein of size 20, one iteration of the second phase takes about 20 minutes. For poly(58-Ala), one iteration already consumes 2.5 hours of CPU-time. For 250 iterations, working on 61 processors, this accounts for about 2 days[6] of computation. As we can see from table 5.10, the number of function evaluations is roughly the same for poly(20-Ala) and poly(58-Ala). Therefore, the number of function evaluations is not the cause for the increase in the cost of the second phase.

If we examine the algorithm closely, it becomes clear that the main reason for an iteration to become more expensive as the protein increases, is the cost of a function evaluation. A function evaluation is $O(n^2)$, n being the number of atoms in the protein. Poly(20-Ala) has approximately 200 atoms, while poly(58-Ala) has almost 600 atoms, accounting for most of the more than eight-fold increase in cost per iteration for the second phase. The rest of the increase in cost can be accounted to the local minimization step. This step requires some linear algebra, which is of order $O(n^2)$ as well. Therefore, it effects the cost per iteration as well, although its contribution is less than the contribution from the more expensive function evaluation.

[5] As this thesis is written, experiments run.

[6] The 250 iterations, each costing 2.5 hours of CPU-time, are executed by 15 groups of processes. Each group consists of a foreman and 3 workers, and roughly executes one iteration of the second phase. Therefore, the total cost can be approximated by the following estimate: $(250 * 2.5)/15 \approx 42$ hours, which is about 2 days.

Chapter 6

Conclusions And Future Work

This thesis has been concerned with the development of a global optimization approach to solve the protein folding problem. We have taken the approach used to solve the Lennard-Jones and the Water problem, and adapted it to solve the protein folding problem. We needed to take account of the loss of partial separability of the potential energy function, as well as of the loss of the expansion step that was present in the original Water algorithm. Among other things, an almost completely new phase 1 of the algorithm needed to be designed. Furthermore, heuristics to select a subset of the free dihedral angles of a configuration needed to be designed and tested.

We have applied the new algorithm to various proteins of the type poly(L-Ala). Up to a protein consisting of 40 amino acids (76 free variables), our algorithm seems to be able to find the global minimum —and therefore the native state— of the protein. For the protein consisting of 58 amino acids (112 free variables), the intermediate results presented in chapter 5 show that the conformation that our algorithm found resembles the native state already closely.

Proteins consisting of 40 or 58 amino acids have not been worked on by that many researchers yet. Especially in the context of global optimization, the ability to solve proteins of these sizes is considered to be quite an achievement. Despite our success in this area, our algorithm is not ready yet to find the native state of real-world proteins. Much more research needs to be done, since real-world proteins consist of $1,000$ to $10,000$ amino acids, which is many more than the proteins we used consist of. Several promising approaches —each an extension of the current algorithm— have been proposed, and some are already in the initial stages of research. If we analyze the cost of the algorithm closely, it turns out that its most expensive part —for the protein folding problem— is the potential energy function evaluation, which is $O(n^2)$ (n being the number of atoms in the protein). For example, a protein consisting of 58 amino acids consists of almost 600 atoms, which accounts for a very expensive function evaluation, and therefore a slow execution of the algorithm. Thus, the proposed new approaches are all based on the idea to have a *cheaper* potential energy function evaluation incorporated in the algorithm.

A first approach is to use a cheap evaluation strategy in the small scale global optimization step of the algorithm. If the dihedral angles are selected in a consecutive way, a local potential energy function can be defined as being the potential energy within the stretch of atoms covered by these dihedral angles. We can either include interactions between the stretch of atoms and the other atoms of the protein, or choose to discard these interactions. In either case, the evaluation of the local potential energy function is considerably cheaper than the full function evaluation. This is because the interactions among the atoms *not* in the stretch are not included, and those interactions account for most of the cost of a full function evaluation. We have done preliminary tests using this approach and the results encourage us to continue pursuing this approach.

A second approach is to use *smoothing* [CSW91] [CSW94]. The original potential energy function contains many local minima. Smoothing tries to limit the search space by defining a sequence of functions, each function more 'smooth' than the previous one. The last defined function should have very few local minima, therefore it should be easy to find the global minimum of this function. Now, the process is reversed. Using the minimizers of a smoother function as starting

points, a less smooth function is globally minimized. Hopefully the minimizers of the smoother function are in the same regions as the good minimizers of the less smooth function, leading to a relatively simple global minimization problem for the less smooth function. Continuing this process, eventually the original function will be minimized. If the sequence of smooth functions was designed correctly, this should again be a relatively simple global optimization problem, and the global minimum of the original energy function should be found. This approach sounds appealing. However, finding a good sequence of smooth functions turns out to be a non-trivial problem, and research in this area is very much going on. Currently we are working on applying smoothing to the Lennard-Jones problem. Learning experiences from this research should help us to use this approach on the protein folding problem.

A third approach is to group the atoms in between the dihedral angles that were selected in step 2b of the algorithm. If somehow a representation for the energy in that group can be found, then the evaluation of the potential energy can be approached by defining a potential energy function based on the groups. This allows for a very cheap small scale global optimization step, in fact, the small scale global optimization step will not be dependent on the number of atoms, but of the number of dihedral angles selected. This feature is very important, since it allows for a small scale global optimization that is constant in time for all size proteins, as long as the number of dihedral angles selected is the same. Therefore, we should be able to execute many more iterations of the second phase, which would allow us to use our algorithm on larger proteins. However, finding such a functional representation for a group of atoms is not easy. The most important fact that must be covered by such a function is the interaction energy *between* the groups that are formed. Since this force guides the folding process, a function that models the interaction energy between the groups accurately is a necessity. Research pursuing this approach has not started yet, but seems to be unavoidable in the future if we want to solve the protein folding problem for real-world proteins.

As a concluding remark we should note that the approaches sketched in the above paragraphs are not necessarily mutually exclusive. It is very well possible to combine, for example, the last two approaches. In the future it might even be unavoidable to pursue such an approach if we want to solve real-world problems.

Bibliography

[AA92] Ruben Abagyan and Patrick Argos. Optimal Protocol and Trajectory Visualization for Conformational Searches of Peptides and Proteins. *Journal of Molecular Biology*, 225:519–532, 1992.

[Aba93] Ruben A. Abagyan. Towards Protein Folding by Global Energy Minimization. *FEBS Letters*, 325(1,2):17–22, June 1993.

[AG94] George S. Almasi and Allan Gottlieb. *Highly Parallel Computing.* Benjamin/Cummings Publishing Company, Redwood City, CA, 2nd edition, 1994.

[Anf73] Christian B. Anfinsen. Principles that Govern the Folding of Protein Chains. *Science*, 181(4096):223–230, July 1973.

[Bal89] Robert L. Baldwin. How Does Protein Folding Get Started? *Trends in Biochemical Sciences*, 14:291–294, July 1989.

[BBO+83] Bernard R. Brooks, Robert E. Bruccoleri, Barry D. Olafson, David J. States, S. Swaminathan, and Martin Karplus. A Program for Macromolecular Energy, Minimization, and Dynamics Calculations. *Journal of Computational Chemistry*, 4(2):187–217, 1983.

[BDE+93] Richard H. Byrd, Thomas Derby, Elizabeth Eskow, Klaas P.B. Oldenkamp, and Robert B. Schnabel. A New Stochastic/Perturbation Method for Large-Scale Global Optimization and its Application to Water Cluster Problems. Technical Report CU-CS-652-93, University of Colorado at Boulder, Department of Computer Science, May 1993.

[BES92] Richard H. Byrd, Elizabeth Eskow, and Robert B. Schnabel. A New Large-Scale Global Optimization Method and its Application to Lennard-Jones Problems. Technical Report CU-CS-630-92, University of Colorado at Boulder, Department of Computer Science, November 1992.

[BG85] W. Braun and N. Gō. Calculation of Protein Conformations by Proton-Proton Distance Constraints: A New Efficient Algorithm. *Journal of Molecular Biology*, 186:611–626, 1985.

[BGOV93] J. Bascle, T. Garel, H. Orland, and B. Velikson. Biasing a Monte Carlo Chain Growth Method with Ramachandran's Plot: Application to Twenty-L-Alanine. *Biopolymers*, 33:1843–1849, 1993.

[BHW87] M. Billeter, T.F. Havel, and K. Wüthrich. The Ellisoid Algorithm as a Method for the Determination of Polypeptide Conformations from Experimental Distance Constraints and Energy Minimization. *Journal of Computational Chemistry*, 8(2):132–141, 1987.

[CD93] Hue Sun Chan and Ken A. Dill. The Protein Folding Problem. *Physics Today*, pages 24–32, February 1993.

[Cor91] Kendall Square Research Corporation. KSR1 Principles of Operation, Revision 5.5, October 1991.

[Cor92] Kendall Square Research Corporation. Technical Summary, 1992.

[Cre88] Thomas E. Creighton. Toward a Better Understanding of Protein Folding Pathways. *Proceedings National Academic Science USA*, 85:5082–5086, July 1988.

[CSW91] T.F. Coleman, D. Shalloway, and Z. Wu. Isotropic Effective Energy Simulated Annealing Searches for Low Energy Molecular Cluster States. Technical Report CTC92TR113, Cornell University, Advanced Computing Research Institute, 1991.

[CSW94] Thomas Coleman, David Shalloway, and Zhijun Wu. A Parallel Build-Up Algorithm for Global Energy Minimizations of Molecular Clusters Using Effective Energy Simulated Annelaing. *Journal of Global Optimization*, 1994.

[DS83] J.E. Dennis, Jr. and Robert B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations.* Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1983.

[Fas89] Gerald D. Fasman. The Development of the Prediction of Protein Structure. In Gerald D. Fasman, editor, *Prediction of Protein Structure and the Principles of Protein Conformation*, chapter 6, pages 193–301. Plenum Press, New York, 1989.

[GL88] Les Goldschlager and Andrew Lister. *Computer Science: A Modern Introduction.* Prentice Hall International (UK) Ltd, Englewood Cliffs, NJ, second edition, 1988.

[GLSW92] Richard A. Goldstein, Zaida A. Luthey-Schulten, and Peter G. Wolynes. Optimal Protein-Folding Codes from Spin-Glass Theory. *Proceedings National Academic Science USA*, 89:4918–4922, June 1992.

[HD85] Stephen C. Harrison and Richard Durbin. Is there a Single Pathway for the Folding of a Polypeptide Chain? *Proceedings National Academic Science USA*, 82:4028–4030, June 1985.

[HGS92] Teresa Head-Gordon and Frank H. Stillinger. Enthalpy of Knotted Polypeptides. *Journal of Physical Chemistry*, 96:7792–7796, 1992.

[HGSA91] Teresa Head-Gordon, Frank H. Stillinger, and Julio Arrecis. A Strategy for Finding Classes of Minima on a Hypersurface: Implications for Approaches to the Protein Folding Problem. *Proceedings National Academic Science USA*, 88:11076–11080, December 1991.

[HK89] L. Howard Holley and Martin Karplus. Protein Secondary Structure Prediction With a Neural Network. *Proceedings National Academic Science USA*, 86:152–156, January 1989.

[KCL90] D.G. Kneller, F.E. Cohen, and R. Langridge. Improvements in Protein Secondary Structure Prediction by an Enhanced Neural Network. *Journal of Molecular Biology*, 214:171–182, 1990.

[KGV83] S. Kirkpatrick, C.D. Gelatt, Jr., and M. Vecchi. *Science*, 220:671–680, 1983.

[KT89] A.H.G. Rinnooy Kan and G.T. Timmer. Global Optimization. In G.L. Nemhauser, A.H.J. Rinnooy Kan, and M.J. Todd, editors, *Handbooks in Operations Research and Management Science, Volume I: Optimization*, pages 631–662. North-Holland, 1989.

[LER92] Ted G. Lewis and Hesham El-Rewini. *Introduction to Parallel Computing.* Prentice-Hall, Englewood Cliffs, NJ, 1992.

[Lev83] Michael Levitt. Protein Folding by Restrained Energy Minimization and Molecular Dynamics. *Journal of Molecular Biology*, 170:723–764, 1983.

[LM92] Scott M. Le Grand and Kenneth M. Merz, Jr. The Application of the Genetic Algorithm to the Minimization of Potential Energy Functions. *Journal of Global Optimization*, 1992.

[LS87] Zhenqin Li and Harold A. Scheraga. Monte Carlo-Minimization Approach to the Multiple-Minima Problem in Protein Folding. *Proceedings National Academic Science USA*, 84:6611–6615, October 1987.

[LS88] Mark Lipton and W. Clark Still. The Multiple Minimum Problem in Molecular Modeling. Tree Searching Internal Coordinate Conformational Space. *Journal of Computational Chemistry*, 9(4):343–355, 1988.

[MMBS75] F.A. Momany, R.F. McGuire, A.W. Burgess, and H.A. Scheraga. Energy Parameters in Polypeptides. VII. Geometric Parameters, Partial Atomic Charges, Nonbonded Interactions, Hydrogen Bond Interactions, and Intrinsic Torsional Potentials for the Naturally Occurring Amino Acids. *Journal of Physical Chemistry*, 79(22):2361–2381, 1975.

[Nor87] J.A. Northby. Structure and Binding of Lennard-Jones Clusters: $13 \leq n \leq 147$. *Journal of Chemical Physics*, 87:6166–6178, 1987.

[NPS83] George Némethy, Marcia S. Pottle, and Harold A. Scheraga. Energy Parameters in Polypeptides. 9. Updating of Geometrical Parameters, Nonbonded Interactions, and Hydrogen Bond Interactions for the Naturally Occuring Amino Acids. *Journal of Physical Chemistry*, 87:1883–1887, 1983.

[Old93] Bart Oldenkamp. Parallel Global Optimization: Applications to Molecular Configuration Problems. Master's thesis, Erasmus Universiteit Rotterdam, August 1993.

[OP92] Eamonn M. O'Toole and Athanassios Z. Panagiotopoulos. Monte Carlo Simulation of Folding Transitions of Simple Model Proteins Using a Chain Growth Algorithm. *Journal of Chemical Physics*, 97(11):8644–8652, December 1992.

[P4] P4. The complete distribution of P4 can be obtained by anonymous ftp from info.mcs.anlk.gov, in the file p4–1.3.tar.Z in pub/p4.

[PF89] Peter Prevelige, Jr. and Gerald D. Fasman. Chou-Fasman Prediction of the Secondary Structure of Proteins. In Gerald D. Fasman, editor, *Prediction of Protein Structure and the Principles of Protein Conformation*, chapter 9, pages 391–416. Plenum Press, New York, 1989.

[PS87] Enrico O. Purisima and Harold A. Scheraga. An Approach to the Multiple-Minima Problem in Protein Folding by Relaxing Dimensionality. *Journal of Molecular Biology*, 196:697–709, 1987.

[PSX94] Panos M. Pardalos, David Shalloway, and Guoliang Xue. Optimization Methods for Computing Global Minima of Nonconvex Potential Energy Functions. *Journal of Global Optimization*, 1994.

[QS88] Ning Qian and Terrence J. Sejnowski. Predicting the Secondary Structure of Globular Proteins Using Neural Network Models. *Journal of Molecular Biology*, 202:865–884, 1988.

[Ric91] Frederic M. Richards. The Protein Folding Problem. *Scientific American*, pages 54–63, January 1991.

[SES89] Sharon Smith, Elizabeth Eskow, and Robert B. Schnabel. Adaptive, Asynchronous Stochastic Global Optimization Algorithms for Sequential and Parallel Computing. Technical Report CU-CS-449-89, University of Colorado at Boulder, Department of Computer Science, October 1989.

[SK90] Jeffrey Skolnick and Andrzej Kolinski. Simulations of the Folding of a Globular Protein. *Science*, 250:1121–1125, November 1990.

[SK91] Jeffrey Skolnick and Andrzej Kolinski. Dynamic Monte Carlo Simulations of a New Lattice Model of Globular Protein Folding, Structure and Dynamics. *Journal of Molecular Biology*, 221:499–531, 1991.

[SLX92] Paul Stolorz, Alan Lapedes, and Yuan Xia. Predicting Protein Secondary Structure Using Neural Net and Statistical Methods. *Journal of Molecular Biology*, 225:363–377, 1992.

[SS92] Sharon L. Smith and Robert B. Schnabel. Dynamic Scheduling Strategies for an Adaptive, Asynchronous Parallel Global Optimization Algorithm. Technical Report CU-CS-625-92, University of Colorado at Boulder, Department of Computer Science, November 1992.

[VD90] Sandor Vajda and Charles Delisi. Determining Minimum Energy Conformations of Polypeptides by Dynamic Programming. *Biopolymers*, 29:1755–1772, 1990.

[Vel] B. Velikson. Structural and Ultrametric Properties of Twenty(L-Alanine). Supplied to us by Ruth Pachter.

[VS85] Max Vásquez and Harold A. Scheraga. Use of Buildup and Energy-Minimization Procedures to Compute Low-Energy Structures of the Backbone of Enkephalin. *Biopolymers*, 24:1437–1447, 1985.

[WC90] Stephen R. Wilson and Weili Cui. Applications of Simulated Annealing to Peptides. *Biopolymers*, 29:225–235, 1990.

[WCMS88] Stephen R. Wilson, Weili Cui, Jules W. Moskowitz, and Kevin E. Schmidt. Conformational Analysis of Flexible Molecules: Location of the Global Minimum Energy Conformation by the Simulated Annealing Method. *Tetrahedron Letters*, 29(35):4373–4376, 1988.

[WKNC86] Scott J. Weiner, Peter A. Kollman, Dzung T. Nguyen, and David A. Case. An All Atom Force Field for Simulations of Proteins and Nucleic Acids. *Journal of Computational Chemistry*, 7(2):230–252, 1986.

[Xue94] Guoliang Xue. Molecular Conformation on the CM-5 by Parallel Two-Level Simulated Annealing. *Journal of Global Optimization*, 1994.