Modified crossover operators for protein folding simulation with genetic algorithms

By
David Jackson

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

Ottawa-Carleton Institute for Computer Science
School of Computer Science
Carleton University
Ottawa, Ontario

September 23, 2004

© Copyright
2004, David Jackson

Library and
Archives Canada

Bibliothèque et
Archives Canada

Published Heritage
Branch

Direction du
Patrimoine de l'édition

395 Wellington Street
Ottawa ON K1A 0N4
Canada

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

ISBN: 0-612-97454-5

Our file Notre référence

ISBN: 0-612-97454-5

The author has granted a non-exclusive license allowing the Library and Archives Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque et Archives Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

In compliance with the Canadian Privacy Act some supporting forms may have been removed from this thesis.

Conformément à la loi canadienne sur la protection de la vie privée, quelques formulaires secondaires ont été enlevés de cette thèse.

While these forms may be included in the document page count, their removal does not represent any loss of content from the thesis.

Bien que ces formulaires aient inclus dans la pagination, il n'y aura aucun contenu manquant.

Canada

PAGINATION ERROR.

ERREUR DE PAGINATION.

TEXT COMPLETE.

LE TEXTE EST COMPLET.

Abstract

In this thesis we cover the application of genetic algorithms to the protein folding problem, which is a very important problem in biochemistry with applications to medicine and drug development. We present two new ideas for improved crossover operators for the folding problem which use secondary structure information from the running population to weight the choice of crossover points. The operators were implemented and tested with a pre-existing protein folding simulator on a number of sequences and the resulting protein structures were compared to the experimentally determined structures for those sequences. They produced structures with 14% improvement in structural similarity for one operator and 16% improvement for the other on average.

Acknowledgements

I would like to thank my supervisors Dr. Jörg-Rüdiger Sack and Dr. Doron Nussbaum for their guidance and financial support, and Dr. James Cheetham and Dr. Gino DiLabio (NRC) for their advice and assistance on biochemistry matters. I would also like to thank HPCVL, Sun Microsystems, Stantive Solutions and NSERC for their contribution of computing resources and funding. I also greatly appreciated Jan Pedersen's patient help with his protein folding software and the help that Ji Chen and Andras Frankel gave me with proofreading my thesis.

For moral support I would like thank my parents and all my friends.

Contents

1	Introduction	1
1.1	The protein folding problem	1
1.1.1	Introduction to Protein Structure	3
1.1.2	Protein Structure Prediction	7
1.2	Thesis Organization	8
2	Survey of Previous work	9
2.1	Ab initio protein folding	9
2.2	Global optimization for ab initio protein folding	10
2.2.1	Combinatorial optimization	11
2.2.2	Simulated Annealing	12
2.2.3	Smoothing/Deformation methods	12
2.2.4	Branch and Bound	13
2.3	Genetic algorithms	14
2.3.1	History	14
2.3.2	Applications of genetic algorithms	16
2.3.3	Elements of genetic algorithms	17
2.4	Using GAs for the protein folding problem	20
2.4.1	Why Use a GA?	20
2.5	Protein folding with GAs	22
2.5.1	Simplified models	22
2.5.2	More detailed models	23
2.6	Non-uniform crossover point selection	29

3	Methods	32
3.1	Design Considerations	33
3.2	Approaches to the problem	34
3.3	Operator C1	37
3.4	Operator C2	39
3.4.1	Preservation of Tertiary Structure	42
4	Implementation and Experimental Results	44
4.1	Implementation details	44
4.2	Experiments	46
4.2.1	Structure of Experiments	47
4.2.2	Performance Measure	48
4.2.3	Diversity Measurement	49
4.2.4	Test Environment	50
4.2.5	Pre-existing Parameters	50
4.2.6	C1 and C2 Parameters	52
4.2.7	Test sequences	59
4.3	Experimental Results	61
4.3.1	Overview	61
4.3.2	Analysis of Results	62
5	Conclusion	72
5.1	Future work	72
	Appendices	75
	A Definition of Root Mean Square Deviation	75
	B RMSD and Energy results	77
	References	84

Chapter 1

Introduction

In this thesis, we present ideas on how to improve the performance of genetic algorithms for the application of protein folding. Protein folding is an important problem with applications to molecular biology, medical sciences and the pharmaceutical industry. In this chapter we cover the basics of the protein folding problem, how genetic algorithms have been applied to the problem, and focus on how the algorithms can be improved by modifications to the crossover operator, as demonstrated by the experimental results.

1.1 The protein folding problem

The protein folding problem is very important and has been identified as a National Grand Challenge in biology (Committee on Physical, Mathematical and Engineering Sciences, 1992) in the United States. Proteins are said to be the building blocks of life. Acting like tiny machines, each type of protein has its own function in the processes necessary for life in an organism. For example, there are many types of enzymes which act as catalysts which accelerate most of the chemical reactions necessary for life. Hemoglobin is an example of a transport protein found in red blood cells which carries oxygen from the lungs throughout the body. Other classes of proteins include:

storage proteins, proteins of contractile systems (e.g. muscle), antibodies (important in the immune system), hormones (e.g. insulin, growth hormone), and structural proteins (e.g. collagen in connective tissue and α -keratin in hair) [78].

It is estimated that there are as many as 100 000 different types of proteins in the human body [78], although many of them bear a strong resemblance to each other. The first protein structure to be fully understood was hemoglobin, which was solved in the 1960's. Today the number of proteins with known structure has grown to a few thousand, yet the amino acid sequences are known for a much larger number. The Human Genome Project is complete and after the DNA sequences are fully decoded they should yield amino acid sequences for every protein in the human body.

Since the 1960's biochemists have been searching for a way to predict protein structure from amino acid sequence, but the problem is still far from solved. Determining the structure of proteins is necessary in order to determine their function, which can help explain biological processes, or be used in drug discovery. Structure determination by experimental methods (using x-ray crystallography and nuclear magnetic resonance techniques (NMR)) is time consuming and expensive. According to a recent estimate [64], for the purpose of drug development, the cost of determining the structure of a target protein is from \$140000USD for a soluble bacterial protein with a success rate of 35%, up to \$2.5million for a human membrane protein target with a success rate of only 10%. This process can take months to several years. Improving the ability to accurately predict protein folds using computer modeling will reduce the need for lab work, saving time and money.

Another closely related problem is the inverse-protein folding problem, where we want to find a sequence which folds into a particular structure. This is also important in drug development, and may be used to design new protein based materials for engineering applications such as information storage for computers, energy conversion and storage, or optical/chemical shielding [45].

Figure 1.1: Chemical structure of an amino acid

1.1.1 Introduction to Protein Structure

Proteins are macromolecules which consist of chains of amino acids. Each different protein has a unique sequence of amino acids. There are 20 types of naturally occurring amino acids used in proteins, each with unique structural and chemical properties. The forces between the amino acids in a protein determine, in part, how it folds into its native structure.

Figure 1.1 shows the chemical composition of one amino acid. In this picture the “R” represents the part which is different for each of the 20 amino acids. They are illustrated in Figure 1.2 with their 3 letter and 1 letter abbreviations. Amino acids are linked together in chains by peptide bonds. Peptide bonds are essentially planar. The ends of a chain are called the N-terminus and the C-terminus, and sequences are always written left-to-right from the N-terminus to the C-terminus. In Figure 1.3, the peptide bond planes are shown by the gray rectangles. The blue nitrogen and black carbon atoms form the backbone, while the pink balls represent the side chains which are different from one type of amino acid to another. While the peptide bonds are rigid, the carbon atom between the two planes (named C^α) has two bonds which can rotate around their axes. The degree of rotation about these two bonds form the dihedral angles named Ψ (psi) and Φ (phi), and they give a protein most of its conformational freedom to attain different shapes.

The Ψ and Φ angles are not free to take any value, but they are actually interdependent and can only attain certain ranges of values. This is because the atoms take up space and must not collide. Figure 1.1.1 shows a simplification of which combinations are allowable.

Figure 1.2: The 20 naturally occurring amino acids (from <http://www.psrc.usm.edu/macrog/protein.htm>)

Figure 1.3: A section of a protein with angles labeled

Figure 1.4: In this Ramachandran plot white regions show disallowed combinations of Ψ and Φ angles for all but the smallest amino acid (glycine). This figure is from <http://www.cryst.bbk.ac.uk/PPS2/course/section3/rama.gif>

Figure 1.5: (a) alpha helix (b) beta sheet. Hydrogen bonds are shown with orange dotted lines. These images are from: http://www.bmb.psu.edu/faculty/tan/lab/gallery_secstruct.html

Proteins have several levels of structure from primary up to quaternary structure. The primary structure includes the amino acid sequence of the protein, and disulfide bonds which can connect different amino acids together. A particular amino acid at a specific position in the chain is often referred to as a residue. Secondary structure is local structure formed by interactions between residues which are close together in the sequence. Two important types of secondary structure include the alpha helix (in Figure 1.5 (a)), and the beta sheet (in Figure 1.5 (b)). Secondary structures are held together mainly by hydrogen bonds, as shown in the figures. Each type of secondary structure also corresponds to certain combinations of Ψ and Φ angles. In Figure 1.1.1 some of the regions which correspond to particular secondary structures are labeled.

Tertiary structure encompasses the interactions between residues far apart in the sequence (see Figure 1.6 for an example), and the structure of the molecule as a whole. Sections of the chain which fold semi-independently are units of tertiary structure called domains. Quaternary structure describes the packing together of separate protein molecules. In this thesis the focus is on prediction of tertiary structure. More information on the protein folding problem for a general audience can be found in [55] or [10], or for a technical audience in [15].

Figure 1.6: Tertiary structure example: residues 192 and 265 are far apart in sequence, yet they are connected by a hydrogen bond. (used with permission from [11]))

1.1.2 Protein Structure Prediction

In this thesis we focus on tertiary structure prediction, and there are a number of different approaches to this problem. For proteins with high levels of sequence similarity with known proteins, it is possible to use homology modeling. Homology modeling [5] involves aligning the sequence to be folded to a template from a known protein structure with a similar amino acid sequence. Since as sequence similarity between proteins decreases their structural similarity also tends to decrease, other techniques must be used for proteins whose sequences have lower levels of similarity to those with known structure. For sequences with medium levels of similarity another technique called fold recognition can be used. For proteins where the sequence similarity is too low, novel fold recognition (also called *ab initio* prediction) can be used [6]. In this thesis we focus on *ab initio* protein folding.

In *ab initio* folding the search for a correctly folded structure is usually primarily guided by the thermodynamic hypothesis, which asserts that proteins tend to fold to their lowest energy state [28]. The energy is the sum of the potential energy from all of the attractive and repulsive forces between the atoms and the surrounding water molecules and can be estimated using an energy function. *Ab initio* protein folding

involves searching for the correct structure which minimizes the energy function.

The approach in this thesis is to use a genetic algorithm for the search. As will be discussed in the following chapter, a number of authors have already used genetic algorithms for this problem. In this thesis we focus on methods to improve the genetic algorithm when applied to the protein folding problem by modifying the crossover operator. Heuristics based on the identification of emerging secondary structured in individual foldings in the population were used to weight the choice of crossover points. This can improve the searching efficiency of the algorithm by encouraging formation and preservation of secondary structures present in the running population which are likely to be correct.

1.2 Thesis Organization

This thesis is organized as follows:

- Chapter 1 contains the motivation, problem definition and outlines our contribution.
- Chapter 2 covers background information on the techniques and previous work done in this area.
- Chapter 3 covers our methods for improving upon previous genetic algorithm based tertiary structure prediction methods.
- Chapter 4 covers implementation and experimental results.
- Chapter 5 includes the conclusion and ideas for possible future work.

Chapter 2

Survey of Previous work

In this chapter we cover previous work on using genetic algorithms for protein folding, and on crossover operators which use some feedback from the environment for choosing crossover points.

2.1 Ab initio protein folding

As mentioned in the introduction, ab initio protein folding involves searching for the lowest energy structure. This assumption is called the thermodynamic hypothesis. Anfinsen performed experiments in which proteins were repeatedly folded and unfolded, and concluded that proteins fold towards their minimum energy structure. He called this the thermodynamic hypothesis of protein folding [2] since it can be explained in terms of the laws of thermodynamics. A protein molecule has potential energy due to the forces between its own atoms and between itself and its environment. In a system of protein molecules this is in the form of heat energy. The second law of thermodynamics implies that if the molecules have more heat energy than its surrounding environment, the heat energy will flow into its environment. As the molecules lose heat energy (enthalpy) the diversity of states they are found in decreases and therefore their entropy decreases. When the heat energy decreases to the

point where the molecules reach equilibrium with their environment the only state they can attain is their native state and thus they are stable. Proteins found in nature have evolved so that they are stable in the conditions they must function in.

Misfolding occurs in some cases, and is involved in many diseases including Alzheimer's disease, Creutzfeld-Jakob disease and bovine spongiform encephalopathy (BSE). There are some cases where proteins have a lower energy structure than their native structure [28], and the thermodynamic hypothesis does not hold, however in the *ab initio* approach we assume it holds true. Therefore in order to search for the correct structure we can search for the global minimum of the energy. The energy of a particular protein structure is estimated using an energy function. The following section focuses on using global optimization methods to search for the lowest energy structure, but other methods based on *ad hoc* statistical models and heuristics are also popular. For a general survey of methods used for *ab initio* tertiary structure prediction see [7].

2.2 Global optimization for *ab initio* protein folding

Ab initio protein folding is often approached as a global optimization problem which requires a search through a very large space of possible structures in order to try to find the one with the lowest energy. To see why this is true, using a simplified model of a protein molecule [56], suppose that each residue can take one of 7 states. Then in a small protein with 50 residues there are $7^{50} \approx 10^{42}$ states. Even if only a small fraction of these states are attainable after other constraints are applied, direct enumeration of all of these states is obviously not possible, so a search algorithm is needed. Two dimensional versions of the protein folding problem have been shown to be NP-complete [16] [3], as well as a three dimensional model [29], thus a precise efficient algorithm for the problem is unlikely to exist.

In this section we summarize 3 global optimization methods which are useful for

the protein folding problem including combinatorial optimization, simulated annealing, and branch and bound, as covered in a survey [22]. Genetic algorithms are covered in more detail in the following sections.

2.2.1 Combinatorial optimization

Dynamic programming techniques have been applied to the protein folding problem with some success [69] [70] [25]. The process begins with small fragments of one residue and their energies are minimized, then structure fragments of two residues are constructed corresponding to all of the pairs of residues found in the sequence to be folded, and their energies are minimized. Then, from the fragments of two residues, fragments of 4 residues are constructed, and so on. This process continues until a structure for the entire protein has been constructed in a bottom-up fashion. For most cases, a prerequisite for application of dynamic programming techniques is that the problem must exhibit optimal substructure, meaning that an optimal solution must contain optimal solutions to subproblems [13]. Suppose that a protein sequence S to be simulated is the concatenation of sequences S_1 and S_2 ($S = S_1S_2$). If the problem has optimal substructure, then if we find the lowest energy state structure for S_1 , and again for S_2 , then compose them together then we could assume that we have a lowest energy structure for S . Unfortunately that is not strictly the case, since S_1 and S_2 will interact in 3 dimensional space in unforeseen ways. But in [69], an *approximate dynamic programming* algorithm is used where, instead of saving one optimal structure fragment for each subsequence, those within a certain threshold of the lowest energy structure are saved, and then used at the next level up in the algorithm. This algorithm was somewhat successful in predicting tertiary structures of small proteins, but according to [22] its weakness is that it over emphasizes local interactions.

2.2.2 Simulated Annealing

Monte Carlo simulation techniques were developed by physicists for simulation of systems of large numbers of particles. The simulations can involve computing integrals over a very large number of dimensions. To reduce computational expense, instead of trying to do numerical integration over all of the dimensions, the integrals can be computed over a random sampling of configurations. Metropolis et al [43] extended this idea, by saying that instead of choosing a set of configurations randomly, we can choose a series of configurations using the following rule: if the randomly chosen state has a lower energy than the previous one, then accept it, but otherwise, accept it with probability $e^{-\delta E/kT}$ where δE is the change in energy, k is Boltzmann's constant, and T is the temperature. If the energy increases from the previous state then the probability of accepting this state decreases with decreasing temperature. This approach gives one way to implement simulated annealing [36]. The name "annealing" comes from a technique to make large, high quality crystals by lowering the temperature of a molten material very slowly as it approaches the melting point. In simulated annealing the Metropolis rule can be used, but we start at a random state at high temperature, and gradually decrease the temperature of the system until it settles down into a low energy state. This technique has applications for many optimization problems in addition to the protein folding problem.

2.2.3 Smoothing/Deformation methods

These methods use a smoothed version of the function to be optimized, which is traced back to the original function. One method is called the diffusion equation method [51], since it uses the equation from physics which describes diffusion over time. For example, if a tank of water lies over the x axis, with a column of water containing dye at a particular location, the diffusion equation could describe the diffusion of the dye spreading out along the x -axis over time. The parameter t which represents time is added to the energy function to obtain the diffusion equation:

$$\frac{\delta^2 E(x, t)}{\delta x^2} = \frac{\delta E(x, t)}{\delta t} \quad (2.1)$$

At $t = 0$, $E(x, t)$ is the original energy function giving the boundary condition:

$$E(x, 0) = E(x) \quad (2.2)$$

To find the local minimum we can start with a smoothed function at some t , find its global minimum, then work backwards decreasing t and updating the global minimum until $t = 0$ in the original energy function. When $t \rightarrow \infty$ (the situation analogous to when the dye is evenly distributed in the tank) the function becomes unimodal so there is only one local minimum which is also the global minimum.

$$t \rightarrow \infty \Rightarrow \frac{\delta^2 E(x, t)}{\delta x^2} = 0 \quad (2.3)$$

Choosing a large enough initial value of t should also allow finding the global minimum. If the initial value of t is chosen correctly, the initial global minimum is found correctly, and t is decreased slowly enough, then this method finds the correct global minimum at $t = 0$. This method has been applied on protein chains 10-20 residues long. Many other surface deformation methods have been developed involving approximations to improve efficiency.

2.2.4 Branch and Bound

Branch and bound is a deterministic optimization method which is often used for combinatorial optimization problems. The α BB algorithm [1] works by bracketing the global minimum of the energy function with converging lower and upper bounds. The space of parameters is subdivided into boxes, each with its own upper and lower bounds on the energy function. Upper bounds are obtained by energy function evaluations or local minimizations of it. To find a lower bound, quadratic terms are added to the energy function to obtain a convex function which is a lower bound on the energy function within the box. The bounding function has one local minima within

the box which can be easily obtained to get a lower bound of the energy function within the box. If a box's lower bound is below the current upper bound, it will be selected to be partitioned and have its bounds refined, otherwise it cannot contain the global minimum it is eliminated from further consideration. A one dimensional example is shown in figure 2.1. A modified α BB, augmented with heuristics to improve efficiency, is described in [20]. They also include secondary structure restrictions based on secondary structure predictions, and they compared using two different representations for structures. They found that using an distance-space representation instead of torsion angle space was more efficient, a result which may be applicable to GA implementations as well.

2.3 Genetic algorithms

This brief introduction to the principles of genetic algorithms (GA's) is based on [44].

2.3.1 History

During the 1950's and 1960's several groups of researchers came up with the idea of applying the concepts of natural selection and natural genetic variation to a set of candidate solutions, in order to "evolve" a better solution.

Rechenberg developed one of the first attempts in the 1960's which was called Evolution Strategies [53]. His algorithm only used a population of two individuals, a parent, and its offspring which is a mutated version of itself. His work was extended by other researchers in Germany in the 1960's and 1970's. This broader class of algorithms is usually referred to as evolutionary algorithms. One of the main differences between these algorithms and those later developed by John Holland is that they lacked any simulation of sexual reproduction.

In the 1960's and 1970's John Holland developed a different approach [32]. He used a population of "chromosomes", which consisted of strings of bits, each one

Figure 2.1: (i) In the first iteration an upper bound is obtained and the domain is bisected. Lower bound L1 is lower than L2 so it is investigated first. (ii) In iteration 2 L3 is higher than the current upper bound so it is eliminated. (iii) The bound on L2 is lower than the current upper bound so it is investigated in iteration 3. (iv) The bounds on L5 and L6 are above the new upper bound so they are eliminated and L4 is bisected. The lower bound on L8 is equal to the current upper bound, so the bounds have converged and the algorithm terminates. This figure is from [22].

representing a “gene”, and applied the idea of natural selection and a set of genetics inspired operators including mutation, crossover and inversion. Mutation causes random changes in the bits of the chromosome. Crossover exchanges two subparts of chromosomes. This simulates what happens in haploid organisms, which have only one copy of each gene for each function. Holland also developed the theoretical underpinnings of his genetic algorithms to explain why they work.

2.3.2 Applications of genetic algorithms

Genetic algorithms are well-suited for problems which require searching for a good solution within a large space of candidate solutions. They can be easily parallelized, because operations on different individuals can be done independently and simultaneously in parallel. This is important for the protein folding problem, since it is very computationally intensive. They are suited to problems that require adaptive solutions, where an algorithm for an exact solution is not known. In the protein folding problem, the GA folds the structure through a series of discrete states, instead of folding through a pathway of continuous motion guided by Newton’s laws of motion as it would in a molecular dynamics simulation.

Genetic algorithms are part of a relatively recent trend in artificial intelligence called complex adaptive systems. Early artificial intelligence work took the top-down approach of trying to codify all of the rules of intelligence to guide the system towards the correct solution. Later, the limitations of this approach became apparent and researchers tried the bottom-up approach of starting with simple rules and trying to capture the complex behavior that emerges from these rules. When using genetic algorithms, the simple rules of mutation, crossover and natural selection are applied, and often complex solutions arise from the application of these simple rules. In the protein folding problem the algorithm is still simple, but there are many details involved and problems that must be solved (particularly tuning the energy function) in order to get a simulation to produce results in close agreement with experimental results.

2.3.3 Elements of genetic algorithms

Pseudo code for the standard genetic algorithm is shown in Algorithm 2. The algorithm goes through a number of generations. In each generation the population is evaluated. Then reproduction is done by selecting two parents with probability proportional to their fitness, a one-point crossover operator is applied to them in order to produce two children, and the children are mutated and added to the next generation.

Algorithm 2 The standard genetic algorithm

```

 $P \leftarrow$  randomly generated initial population
 $Q \leftarrow \text{emptyset}$ 
for  $i = 1$  to Generations do
  evaluate( $P$ )
  for  $i = 1$  to  $PopulationSize/2$  do
     $p_1 \leftarrow \text{fitnessProportionalSelect}(P)$ 
     $p_2 \leftarrow \text{fitnessProportionalSelect}(P)$ 
     $(q_1, q_2) \leftarrow \text{crossOver}(p_1, p_2)$ 
    mutate( $q_1$ )
    mutate( $q_2$ )
     $Q \leftarrow Q \cup \{q_1, q_2\}$ 
  end for
   $P \leftarrow Q$ 
   $Q \leftarrow \text{emptyset}$ 
end for

```

Representation is an important issue in GA implementations. The chromosome, often represented as a bit string, has a particular meaning for each different application. For example, if we wanted to optimize a function dependent on several values, the inputs to the function would be encoded in the bit string. For protein folding, we can represent a protein structure using the set of bond angles, which can be encoded in a bit string. The choice of representation often has a large impact on the effectiveness and efficiency of the fitness function and the operators. An instance of a chromosome is referred to as a genotype, while the expression or interpretation of this chromosome (as a set of torsion angles, for example) is referred to as the phenotype.

Figure 2.2: mutation operator

The *fitness function* guides our search through the space of candidate solutions. For example, GAs are often used for optimization problems, where the fitness function used is the function to be optimized. For protein folding simulation we want to find the minimum energy state so we use energy as our fitness function.

When we reproduce our population of candidate solutions, we want to select the fittest individuals more often than the less fit in order to push the population as a whole towards a better solution. This is called *fitness proportional selection*.

One algorithm for doing fitness proportional selection is called roulette wheel selection because it could be implemented on a real roulette wheel where each individual is given a slice of the wheel proportional to its fitness. When the wheel is spun to a random location the probability of choosing an individual is proportional to the fraction of its fitness out of the total fitness of the entire population.

The *mutation operator* (illustrated in Figure 2.2) makes random changes to offspring in order to explore different parts of the search space. For the bit string chromosome representation, usually each bit is given a small probability of being flipped. Often, for different representations, other types of mutation operators specific to that implementation are used.

The *crossover operator* exchanges subparts of chromosomes between two parents in order to produce offspring. There are several types of crossover as illustrated in Figure 2.3, some of which work better in some applications than others.

The simplest type is one-point crossover. In this case one crossover point is chosen and the parts of the chromosome before and after it are exchanged. The only problem with one point crossover is that when every point in the chromosome has the same probability of being chosen, the parts in the center of the chromosome are more likely to end up in different children than the parts towards the ends. To see the problem

Figure 2.3: crossover operators: a. one point crossover; b. uniform crossover

with this, suppose that both parents contain different chunks of good genetic material near the ends of a chromosome. Then they are not as likely to be combined together into one child as if they were near the middle. This problem can be alleviated partially by using two-point or k-point crossover. For a larger number of crossover points, we can use uniform crossover where at each point in the chromosome there is a 50% probability of crossover.

The crossover operator makes an attempt to recombine the good parts of the chromosome from each parent by choosing the parts at random. The offspring may receive good “genes” from their parents resulting in higher fitness, in which case the offspring will be selected more often and in the next generation the genes will become more common in the population. Conversely, the crossover operator helps reduce the number of bad genes in the population, since individuals who receive them will be selected less often. The chromosomes may also have epistasis (interdependence of the effects of different genes in the phenotype) and the GA will help to find good combinations of genes that lead to higher fitness, by the action of the crossover operator producing many permutations of the available genes. A more precise theory to explain how this works called the fundamental theorem of genetic algorithms was developed by John Holland [32].

Application of genetic algorithms to a particular problem requires careful consideration of representation, fitness function, and what operators should be used and with what frequency in order to be successful. There has been a great deal of theoretical and experimental work done but solving a particular problem using a genetic algorithm still involves intuition, experience and trial and error.

2.4 Using GAs for the protein folding problem

In this section we talk about why GAs are a good approach to solving the protein folding problem. Surveys on this topic can be found in [48] and [14]. Many different implementations using genetic algorithms for protein folding have been done over the last 15 years, but progress has been slow because of the difficulty in comparing different approaches. It is difficult to decide whether to attribute the success or failure of an approach to search properties of the algorithms or effectiveness of the fitness function. Most authors used different types of energy functions and different variations of the genetic algorithm.

2.4.1 Why Use a GA?

The most realistic method of modeling protein folding is to use Cartesian space molecular dynamics simulations, where all protein and solvent molecules are represented explicitly. Their interactions are all calculated through a complex and computationally expensive force field function. The physics involved is understood well enough for accurate modeling, but unfortunately, for molecules the size of proteins, it is too computationally expensive to run a molecular dynamic simulation for long enough to observe folding. It is currently only possible to run a simulation for a few nanoseconds (10^{-9} s), but in order to observe folding we would need a simulation that runs on the scale of microseconds (10^{-6} s) to milliseconds (10^{-3} s). Molecular dynamics simulations for protein folding require very large computing resources e.g. the Folding at Home project [77], and [66]. The limitation is that the hyper surface of the energy

function is very fine grained, that is, very small changes in atom positions can cause large changes in energy. The simulation must be run with sufficient time resolution to avoid skipping the bumps in the energy surface.

There are two main approaches to reducing the computational cost of folding simulations. First, the descriptions of molecules can be simplified, by reducing the level of description. This simplifies the computation of energy, and also smoothes out the surface. Second, one can take larger steps in the conformational space. We can do this by using an optimization algorithm such as Monte Carlo (MC) simulation (described below) or a GA to minimize the energy function. Conceptually, molecular dynamics is like an optimization algorithm, since the trajectory of the molecule takes the path of steepest descent on the energy surface and seeks the lowest energy structure. In a GA or MC simulation, individuals of the population represent snapshots of possible intermediate conformations and therefore they allow larger time steps and avoid the cost of calculating derivatives associated with molecular dynamics simulations.

GAs were found to be 100-200 times faster than MC simulations for protein folding under certain conditions [67]. However, they can complement each other and using the GA for global optimization with MC for local optimization, or just for generating the initial population, is a good combination. The genetic crossover operator may introduce collisions or other energetically unfavorable interactions, but with a small amount of optimization, a larger proportion of good offspring can be produced. This technique of combining local optimization with a GA is referred to as a Lamarckian genetic algorithm [72], after Lamarck's theory of evolution in which it was believed that offspring could acquire traits that their parents had acquired during their lifetime [54].

2.5 Protein folding with GAs

2.5.1 Simplified models

Early attempts to use GAs for the protein folding used simplified models of proteins for the purpose of evaluating the feasibility of applying GA for the problem. Most used a variation of the hydrophobic-hydrophilic's model of amino acid interaction. The set of 20 natural amino acids can be classified into three groups: those that repel water (hydrophobic), those that have an affinity for water (hydrophilic), and those that are ambivalent. Hydrophobic amino acids are usually found at the core of the molecule where they are shielded from water molecules, whereas hydrophilic amino acids are usually found on the outer surface. A simple fitness function was used which is based on counting the numbers of amino acids found in their preferred environment. The protein is modeled as a chain on a grid, where the grid points represent amino acids. The chain may take a turn in any direction at each grid point. This was done for both 2 dimensional grids [67] [39] and 3 dimensional [68] [46] grids.

Unfortunately, the hydrophobic-hydrophilic model is so simple that it is questionable as to whether it provides any useful predictions about how proteins fold. However, this model has been used as a testing ground for algorithms for protein folding. The first GA approach using this model by Unger and Moult [67] helped to show that GAs are a viable approach to the protein folding problem and inspired further work using more detailed models. Recently other algorithms have been tested hydrophobic-hydrophilic 2D lattice model model including a hybrid GA and tabu search approach [33], an ant colony based approach [63] and systematic crossover [37]. Systematic crossover involves evaluating the children of crossover for every possible crossover point between two parents, and then keeping the best pair of children. It was used along with another technique called pioneer search to prevent premature convergence.

2.5.2 More detailed models

In this section, we examine the works of several research groups who used GAs for protein folding simulation with detailed models.

The first attempt to produce tertiary structure in proteins using a GA was done by Sun [65]. His model used a complete representation of the backbone with one virtual atom per side chain. A potential of mean force fitness function was used, which is based on empirical parameters derived from known protein structures. They used a library of structure fragments 2-5 residues long to construct the initial population and make mutational changes. The fragments were selected based on sequence similarity to the target sequence. So to construct a member of the initial population, they would find a fragment which matches the first 2-5 residues, then find another fragment to match the following 2-5 residues, and so on. A similar procedure was used for the mutation operator.

Bowie and Eisenberg [9] used a similar approach to Sun, but instead of using sequence similarity to match fragments to the sequence, they used their 3D profile method, originally developed for verification of experimentally determined structures. The 3D profile method assigns a score to each residue based on its local environment characterized by polarity, solvent-accessible surface area and secondary structure type. It gives a value which assesses the compatibility of a sequence with its structure. In order to prepare the initial population for the GA, they used the 3D profile method to search for structure fragments with a high level of compatibility with corresponding segments of the sequence of the protein to be folded. This was to find a pool of fragments with a higher degree of structural similarity to the correct folding. The pool was used to construct the initial population for further optimization by an evolutionary algorithm.

A number of implementations used the assumption of knowledge of correct secondary structure (e.g. [17] [18]). This is a valid assumption, since secondary structure prediction techniques are better than 70% accurate in many cases [57]. Recently 76% accuracy was achieved [34]. Dandekar and Argos used secondary structure constraints

and they used Holland's bit string representation where each residue can take one of 7 states (each is a combination of Ψ and Φ angles) which are stored in 3 bits per residue. This makes it a more coarse grained representation than the others, but this allowed them to use the classical bit string representation and operators.

Steffen Schulze-Kremer [61] [60] used a combination of ideas from genetic algorithms and Rechenberg's evolution strategies. Due to the latter influence, he uses a smaller population size and a large number of generations. For representation, he used the main chain torsion angles ψ and ϕ , holding ω constant at 180° , and the set of side chain torsion angles $\chi_1 \dots \chi_n$ for each residue. A library was used for the mutation operator, as in other implementations. It was created by applying the nearest-neighbor clustering algorithm to the set of torsion angles for each amino acid type in a set of 129 proteins. The clustering created a representative set of torsion angles. Torsion angles were selected for the mutation operator with a probability distribution derived from their frequency. In addition to crossover and mutation operators, he added an operator he called *Variate* which makes small local changes to torsion angles. He also used a vector fitness function, where each element represented different components of the fitness function. For that reason he used the Pareto selection algorithm. To select an individual, we start with the entire population and remove individuals one at a time by choosing one of the components of the vector and removing the individual with the worst value for that component. This process continues until we are left with the desired number of individuals. Schulze-Kremer's simulator was successful when using RMS deviation as a component of the fitness function, but without it he could not achieve results with a good level of structural similarity. Therefore he concluded that the problem with his implementation lies in his fitness function.

The implementation by Pedersen and Moulton [50] used a combination of a GA for global optimization and MC simulation for generating the initial population and local optimization within each generation. We cover this implementation in more detail because we have used their program as a basis for testing our improved crossover operator. The overall data flow of their application is illustrated in Figure 2.4.

First the amino acid sequence is used to build a model using GROMOS templates.

Figure 2.4: Data flow diagram for Pedersen and Moult's implementation

The empirically derived templates specify the bond lengths which remain the same throughout the simulation.

Next, the starting conformation for the MC simulation was generated by randomly choosing bond angles from the library of observed ϕ , ψ , and ω values. To prevent clashes, angles are assigned one at a time, checking for clashes, and if one is found, backtracking and reassigning angles. This procedure is also similar to the one discussed in [4]. This algorithm takes a number of steps which vary exponentially with the length of the peptide in the worst case, but in practice collisions are rare and the number of steps is almost linearly proportional to the number of residues in the peptide.

The random starting conformation is used as the starting point for an MC simulation. At each step of the MC simulation one of several types of random changes is applied to the structure. The initial population is created by sampling it every 100 steps. So to generate an initial population of 200 individuals it would run the MC simulation for 20000 steps, sampling every 100 steps. This initial population is used as the input to the GA, and after a certain number of generations the lowest energy member of the population in the final generation is taken as the prediction.

Like most other implementations, Pedersen and Moult used a library of dihedral

Figure 2.5: Steps involved in Pedersen and Moul's crossover operation

angles from experimentally determined protein structures. This was used to bias selected angles towards the experimentally observed distributions in a number of MC and GA procedures. This gives the angles values that are more often found in nature, in order to avoid energetically unfavourable conformations and improve the efficiency of the GA.

The MC simulation uses a mixture of different types of moves. Moves that operate on the main chain include: replacement of single ϕ/ψ angle pairs from the library, small perturbations of triples of adjacent ϕ/ψ angles, and replacement of single ϕ/ψ angle pairs chosen from a representative set of the seven classes of secondary structure (α , β , unstructured coil, extended, pleated β , left handed α , and cis peptide β). Side chain moves include library moves and small random perturbations. The decision to accept a move or not is made using the Metropolis rule. The probability of acceptance is $p = e^{\frac{-\Delta\Delta G_{total}}{kT}}$, where $\Delta\Delta G_{total}$ is the difference in total energy of the conformation between the states before and after the move, k is Boltzmann's constant, and T is the temperature which decreases linearly every generation.

Reproduction in this GA differs significantly from the standard algorithm. Firstly, parents are selected with uniform probability, instead of using fitness proportional selection. However, selection is done after the crossover operation using the Metropolis rule, based on comparing the fitness of the children to the parents. If a child is rejected then its parent is copied to the next generation, otherwise it replaces its parent.

Figure 2.5 shows how the crossover operator works. It is quite different from the standard GA crossover operator. First the parts of the chain on either side of the $C - C^\alpha$ bond between the two parents are exchanged. At the crossover point it does a set of 50 trials for different ψ/ϕ angle pairs, half of which are chosen from the library and half from the same representative set used for the MC simulation. If the best child is better than both parents it is accepted for the next generation. Otherwise it is accepted with the Metropolis rule using the average of the parent's energies: $p^{accept} = e^{\frac{\Delta\Delta G_1 + \Delta\Delta G_2}{2kT}}$, where $\Delta\Delta G_1$ and $\Delta\Delta G_2$ are the differences in energy between the child and each parent. If one child is accepted, we keep the best parent, or if neither child is accepted, we keep both parents. This crossover operator is used to form 90% of the next generation, and the other 10% is formed using elitism, by keeping the 10% of the population with the lowest free energies.

The choice of crossover points is weighted depending on the conformational diversity at each residue in the initial population. Conformational diversity $\sigma_{local,i}$ is defined as the standard deviation of ΔG_{local} at each residue. Then, the probability of choosing crossover point i is $p_i = \frac{\sigma_{local,i}}{\sum_{m=1}^M \sigma_{local,m}}$. This feature was absent from the code provided, but it was not essential since, according to the authors, it only serves to slightly increase the rate of convergence. However the success of this method it gave some evidence to suggest that our new weighted crossover method would also be successful.

The representation used in this GA is also not traditional. A conformation is represented by the coordinates of its atoms stored in 64 bit floating point numbers (C double). This is a convenient representation because coordinates of the atoms are needed for the energy function. The GA actually searches in torsion angle space, because the bond lengths between the atoms remain constant, and only the torsion

angles are modified by the crossover operator. It is easy to convert back and forth between coordinates and torsion angles.

If we were using the traditional bit string representation we would have to hold the bond lengths constant and encode the torsion angles with real number encoding. It is not feasible to use a real number encoded coordinate representation because it does not preserve bond length constraints. Unlike Pedersen and Moults representation, in a real-coded bit string representation is that the crossover points may occur anywhere in the bit string, including between torsion angles within the same residue and within the angles. This representation could be less efficient since the Ψ and Φ angles are highly interdependent and many combinations are disallowed, as shown in Figure 1.1.1. In [40], a coordinate transformation is applied to the angles to preserve physical constraints. It effectively removes most of the disallowed regions of ψ and ϕ angles, increasing the efficiency of the GA by preventing the possibility of generating structures containing those unattainable angles which would otherwise have to be evaluated by the energy function before being eliminated.

In any case, we chose to use their floating-point coordinate representation searching in torsion angle space, since their code was written such that the representation and operators are all very closely coupled so it is difficult to modify and changing representations would have required a complete rewrite.

Pedersen and Moults tested their program on a set of 28 peptides ranging in length from 10 to 14 residues. The peptides used were independently folding units (IFUs), which are subsequences of larger proteins chosen according to certain chemical criterion in order to find structures that fold largely independently from the rest of the structure. The RMS deviation was used to assess the agreement of the predicted structures with experimentally determined structures. For 18 of the 28 IFU's tested, the program produced structures with an RMS deviation of less than 3Å, a threshold below which structural similarity is visually obvious. Populations converged so that the individuals had an RMS deviation of less than 1Å from each other. Pedersen and Moults also noted that their GA found lower local minima in the energy function than previous attempts using MC alone.

2.6 Non-uniform crossover point selection

In the past, a number of different crossover operators have been developed which identify groups of bits to be kept together. This is motivated by the building block hypothesis [32], [23], which states that when sections of a chromosome exhibit higher than average fitness, they will be passed on to the next generation and combined with other blocks with higher than average fitness forming larger blocks and so on, eventually producing highly fit chromosomes. The standard GA does this anyways, but there are a number of ways of accelerating building block formation to improve performance.

One approach, which uses only using feedback from the fitness function, it is called adaptive or self-adaptive crossover. An example of adaptive crossover is the operator developed by Oppacher and White [73]. Each bit of the chromosome is augmented with an automaton. The state of each automaton is mapped to a probability of crossover at that point. The state of the automatons in a child are updated depending on its fitness relative to its parents. If the child is more fit than its mother(father), then the bits that came from the mother(father) are rewarded, and, if the child is less fit than the mother(father), then the bits that came from mother(father) are penalized. A reward corresponds changing the state of the automaton such that it gives a higher crossover probability and a penalty is the reverse. This operator is based on uniform crossover in that, when the automatons are in their initial random state, it is equivalent to uniform crossover.

Another crossover operator called Adaptive Linkage Crossover was developed by Salman et. Al [58]. It is based on the definition of linkage between two genes. Linkage is defined as the probability that a child should inherit a particular pair of genes from the same parent. They also define General Linkage Crossover, which can use problem specific information about linkages. This method is not directly applicable using knowledge of secondary structure, but we present a method (operator C2) that is somewhat similar in the next chapter which uses only the linkages between adjacent genes.

A thorough survey of adaptive crossover operators is given in [71]. In general, those crossover operators examined were similar to uniform crossover. Uniform crossover does not work well for the protein folding problem because when a protein structure becomes compact, and in this case we are mainly dealing with protein sequences which fold into compact globular shapes, there are many more possibilities for collisions between atoms or for other conditions which lead to high energy. To avoid this problem, GA protein folding simulations have used one or two point crossover with some kind of search for a way to join the chains at the crossover point in a favourable way. [49] evaluates the energy of the structure for 50 pairs of ψ/ϕ angle pairs and chooses the one with lowest energy. Without this step, the GA does not converge to a low energy. Similarly in [4], sterical constraints (in other words prevention of collisions between atoms) are checked after the crossover operation and another ψ/ϕ angle pair is chosen at random for the crossover point in case of collisions. It would also be possible to do a more sophisticated search at this stage instead of a random search, in particular a constructive technique such as ant colony optimization may work, but application of this method would require careful consideration of how much structure should be preserved by the crossover operator, since reconstructing the entire structure every time would be destructive and time consuming.

It would be possible to use two or more crossover points, however this would greatly increase the searching necessary in order to ensure good joint angles. For Pedersen and Moult's one-point crossover operator they found that it is necessary to search no less than 50 randomly selected joint angle pairs and select the one with the lowest energy [49]. For two point crossover, it would be necessary to do a search for both joint angles, and the searches are dependent on each other, since together they may cause collisions. Therefore, since it is necessary to evaluate 50 joint angle pairs for one crossover point, if two crossover points were used then it may be necessary to evaluate $50 \times 50 = 2500$ joint angle pairs. While this may be an overestimate, because the segments between each crossover point are smaller and will have simpler interactions, using two or more crossover points will greatly increase the number of structures to be evaluated. Since the energy function evaluation is the most time consuming step of running the GA, using more crossover points should be avoided.

Messy GAs are another technique motivated by the idea of accelerating building block formation. Simple GAs suffer from the problem that the genes that make up the building blocks are at a fixed position in the chromosome, but they must be close to each other to avoid being broken up by the crossover operator. Messy GAs deal with this problem by encoding the gene's value along with its position. This allows tighter linkage between pieces of a building block than with a fixed encoding [27]. In messy GAs chromosomes are variable length so they may be over specified, where several values exist for the same position, or under specified, where not all positions have values. The crossover is replaced by a cut-and-splice operator, and mutation is often not used. So-called "Fast" messy GAs use a more efficient algorithm for generating the initial population to reduce the overall time and space complexity of the algorithm [26].

Fast messy GAs were applied to the protein folding problem in [40]. They used a torsion angle representation where each angle (including the side-chain angles) was encoded with 10 bits. They also used an elegant method of preserving constraints on ψ and ϕ angles by using a coordinate transformation. They included methods of incorporating secondary structure constraints, and they also seeded the initial population with secondary structures. Both methods used information from secondary structure predictions and produced small improvements in the fitness of the resulting structure. They also found that including local optimization (making it a Lamarckian GA) improved the results.

It may be possible to develop one or two point adaptive crossover operators, but for our operators, described in the following chapter, problem-specific secondary structure information is used which is independent of the fitness function.

Chapter 3

Methods

In this chapter we present our two approaches for improving the crossover operator using secondary structure information and then implemented in detail. We focus on using secondary structure information to weight the choice of crossover points. The first operator (C1) gives higher weights to points in unstructured regions of the chain and borders between secondary structure elements. Operator C2 assigns weights using statistics on the occurrence of pairs of secondary structures together in real proteins.

We hypothesize that these approaches will allow a GA based protein folding simulator to produce structures with a higher structural similarity to the experimentally determined structures in comparison to the unmodified crossover operator. The structural similarity is measured using was measured by calculating the root mean square (RMS) deviation (see definition in Appendix A) between the two structures.

Protein folding most likely involves an initial formation of secondary structures, followed by condensation of these elements into tertiary structure. We propose emulation of this behaviour in the GA by biasing the choice of crossover so that it avoids breaking secondary structures present in the current population. The idea is that it will restrict the search space to more favourable areas leading to faster convergence to the result without sacrificing its quality.

3.1 Design Considerations

It is important to choose the correct weights for crossover points. Weighting the choice of crossover points too heavily will cause it to choose the same points too often producing too many similar children. This will decrease the performance of the GA for one of two reasons. Premature convergence could occur if the favoured crossover points also happen to produce individuals which have a better than average fitness. This case does not happen easily with our GA since does not use traditional fitness proportional selection. The other possibility is stagnation, the search is too focused by the narrow choice of crossover points, causing the the progress of the GA to slow down because it takes longer to find fitter individuals. If weights are too low, the crossover operator will perform no better than the unmodified version.

A choice was made to preserve the structured regions by weighting the choice of crossover points, rather than rewarding them through the fitness function. The reason is that a potential problem exists in GAs with rewarding intermediate building blocks disproportionately more than the smaller, lower level ones. In general, rewarding intermediate building blocks more than smaller ones can lead to a phenomenon called *hitchhiking* [23]. Forrest and Mitchell showed that parts of the chromosome with low fitness can “hitchhike” along with the highly fit parts and cause premature convergence. In our approach where we use secondary structure information, secondary structure elements could be identified and given higher values in the fitness function. This could lead to the hitchhiking problem because secondary structures must be built up from smaller pieces in the population and we do not have prior knowledge about how many residues should be included in a secondary structure element. For example, suppose that a protein sequence when correctly folded should contain a secondary structure element of type t and length l . If secondary structure elements of type t and length less than l are given a reward in the fitness function which is disproportionate to their length, it may prevent the GA from finding the correct structure of length l . For this reason, our algorithm includes modifications to the crossover operator instead of the fitness function.

3.2 Approaches to the problem

Using the general idea of weighting the choice of crossover points we came up with a number of different approaches. The first idea was to look for incorrect secondary structures in a protein and try to bias the choice of crossover points so that they are broken up. We would like to have a method of examining the patterns in a small window of residues at a time and assigning a probability of error within that window. The window would move along the entire length of the chain so that each residue in the chain receives a probability of error. Performing a crossover operation in the vicinity of a point with a higher probability of error would be more likely to remove the unfavourable structure within its window. We examined using two programs which can assess the quality of a structure. ERRAT [12] examines the patterns of non-bonded interactions between the residues within a sliding window certain size, and any other residues close enough to interact with them. Another method called 3DProfiles[41] examines patterns in 3 different environmental conditions for a window of 3 residues.

Both of these methods evaluate the compatibility of a sequence with a structure and are mainly used for checking experimentally determined structures for errors. Unfortunately, they were not appropriate for our needs because although they assign an error value to each residue, the error value takes into account the interactions between the residues in the window and every other residue in the chain, so it does not indicate the probability of a local error. As an example where using ERRAT or 3DProfiles will fail, suppose we have a structure with three high error values at positions i , j , and k in the chain. From this information it is not possible to tell whether the high error value at i is caused by i interacting with j unfavourably, or i with k , or residues close to i in the chain (within the sliding window) interacting with each other. Originally we had hoped that we could find the probability of error due to the latter case only, but we were unable to find such a method. This ambiguity prevents us from using ERRAT or 3DProfiles for the purpose of finding good crossover points.

Perhaps the methods of ERRAT or 3D profiles could be modified to eliminate the ambiguity and still use information from residues which are not close to each other in the chain. A graph could be constructed where the vertices represent the atoms, and edges are given weights depending how favourable their interactions are. We could choose a crossover point such that it tends leave the maximum number of favourable interactions in tact while the maximum number of unfavourable ones are cut. This idea is inspired by domain partitioning methods explained further in Section 3.4.1, however the implementation is beyond the scope of this thesis.

We came up with several ideas for weighting the crossover points based on secondary structure. The question is, using the information derived by identifying secondary structure in both individuals involved in the crossover, how can it help in choosing a crossover point which is likely to give the best child? Do certain types of secondary structure elements tend to be longer than others? Are there certain patterns of combinations of secondary structure elements which will be found more often than others?

A simple approach, which does not involve any specific knowledge about the properties of different structure types, is to use the weighting of crossover points to encourage the formation of larger secondary structure elements from smaller ones and preserve structure elements present in the initial population. We call this operator *C1* and it is covered in detail in Section 3.3.

Another idea to use structure information in the crossover operator is to investigate the frequency of adjacency of different types of structure elements and choose a crossover point which will produce a more favourable pairing. We call this operator *C2*. For example, perhaps a helix is rarely directly adjacent to a beta turn, but it is often adjacent to a coil, therefore HC or CH are preferred over TH or HT. Then suppose we are running a simulation on a peptide with 12 residues and we want to choose the crossover point for two individuals with the following secondary structures:

CCTTTT|HHHHHC

CCCCC|CHHHHC

Then, in order to get rid of the unfavourable TH, we could choose the 6th crossover point and swap the tails of the parents to produce the following more favourable structures in the children:

CCTTTT|CHHHHC

CCCCC|HHHHHC

To encourage the choice of the 6th crossover point it should be given a higher weight. This idea is explained further in Section 3.4. Note that both C1 and C2 are weighted forms of one point crossover, but they could easily be extended to two point or k -point crossover.

We also briefly considered several other schemes. One was to use dynamic programming as a local optimization step to find the best possible children that can be produced from two parents in the crossover operator. This idea has been applied in permutation problems [76]. A dynamic programming algorithm similar to the one described in Section 2.2.1 could be used. We decided not to pursue this idea because the dynamic programming technique for protein folding simulation appears to have been only partially successful in [69].

We also considered using some type of *hierarchical crossover* operator. Hierarchical crossover is based on the operator used in Koza’s Genetic Programming [38], in that it preserves the semantics of a program represented as a tree, but can be applied to more general problems where the chromosome is variable length but has hierarchical structure which must be conserved in order for it to make sense. Protein structure can be described in terms of a hierarchy, since secondary structure elements are composed of a sequence, and tertiary structures are composed of a number of secondary structure elements. However, the chromosomes are not normally variable length and, more importantly, there are no hard and fast rules of composition and they would be difficult to define. Perhaps “fuzzy” rules could be developed, but we did not pursue this idea any further.

3.3 Operator C1

In this section we propose a new crossover operator which is modified so that it chooses crossover points in a way that preserves secondary structure elements. This operator is intended to improve the performance of the folding simulator by both encouraging the formation of larger structures out of smaller units, and preserving structures present in the initial population. In order to do this, it identifies the secondary structure elements in each individual, and weights the selection of crossover points towards unstructured regions and the borders points between structures of different types.

In order to implement such a crossover operator, structure elements must be identified for each individual. The structures are different for every individual in the population. To test our ideas by manually identifying structure would require going through hundreds of individuals for a short run of the GA, so automated methods are needed.

There are a number of well established methods for identification of secondary structure and software is freely available including DSSP [35], and Stride [24]. In general, these programs assign a type of secondary structure to each residue in the chain.

Although there are a number of different software programs using different methods for secondary structure identification, most of them give very similar results [8]. In this thesis Stride was used because the source code was easy to obtain. Stride returns a string representing the secondary structure where each character represents the type of secondary structure of each residue. The characters are: C, coil/unstructured; H, α -helix; B, β -bridge; E, extended strand; G, 3-helix; I, 5-helix; T, turn; S, bend.

Algorithm 3 shows how we can obtain weights for each potential crossover point. Before crossover operations begin we would run a function to obtain crossover weights for each individual conformation, based on the output from a secondary structure determination program. In order to avoid breaking secondary structure, we give the

the unstructured (coil) regions and the borders between different types of structure a higher weight. Then during a crossover operation the weights from two parents are combined to obtain the weights for that pair.

The `getSecondaryStructure()` function runs the secondary structure identification subroutine or external program and returns an array containing the type of secondary structure for each residue. In the code below, all structured regions receive secondary-Weight as their weight where $0 < \text{secondaryWeight} < 1$. It would also be possible to use different weights for different types of structure (alpha helix, beta sheets, beta turns, etc) if further analysis determined that they should be different. Lastly, the weights are normalized in order to obtain probabilities. If roulette wheel selection is used, the normalization step can be omitted since it is part of that algorithm.

Algorithm 3 `getProbabilities(individual)`

```

r ← numberOfResidues(individual)
for i = 1 to r do
  if i > 0 and type[i] ≠ type[i - 1] or (i < r - 1 and type[i] ≠ type[i + 1]) then
    prob[i] ← BORDER_WEIGHT
  else if type[i] = COIL then
    prob[i] ← COIL_WEIGHT
  else
    prob[i] ← DEFAULT_WEIGHT
  end if
end for

```

`BORDER_WEIGHT` is the weight to assign the two crossover points on either side of a point where the type of structure changes from one type to another, and `COIL_WEIGHT` is the weight to assign to a coil segment.

In order to calculate the probability that the same crossover point is chosen on both chains the probabilities from the two parents are multiplied since the two sets of probabilities are assumed to be independent. From this result we use the roulette wheel selection algorithm to select a crossover point. Normalization is not necessary because `rouletteWheelSelect()` does this step. Algorithm 3.3 combines the probabilities from two parents and selects a crossover point:

Algorithm 4 chooseCrossoverPoint(parentProbabilities1, parentProbabilities2)

```

for  $i = 1$  to  $size(probabilities1)$  do
     $probabilities[i] \leftarrow parentProbabilities1[i] * parentProbabilities2[i]$ 
end for
return  $1 + rouletteWheelSelect(probabilities)$ 

```

3.4 Operator C2

In this crossover operator is the crossover points are weighted according to the frequency of each pair of secondary structures occurring adjacent to each other in real proteins. This idea is somewhat similar to the General Linkage Crossover operator defined in [58] and summarized in Chapter 2, however [58] use linkage information between every pair of genes in a chromosome, while our operator only uses the linkage information from adjacent genes in the chromosome, since secondary structure information can only give us information about local linkages. Since the unfavourable structures usually have higher energies anyways, they would eventually be eliminated by selection, but we hope that this approach can improve the efficiency of the GA by causing it to converge to a higher quality result in fewer generations.

A simple way of implementing this idea is to look at all the statistics on pairs of structure types occurring adjacent to each other in a library of experimentally determined protein structures. We can use this library to find statistics on how often pairs of structure types occur together. This would give a two dimensional matrix with a frequency for each pair. One would expect that these statistics would be dependent on the choice of proteins analyzed to obtain them. Perhaps using different sets of statistics tailored to the type of protein being analyzed would be appropriate. A general set of statistics which should be appropriate for most sequences was obtained by examining the secondary structures in the entire RCSB Protein Data Bank [52]. There were 26531 structure files in the data bank, and of those files 24788 of them were successfully examined with Stride. Failure on some of the files was most likely due to files containing non-standard (synthetic) amino acids, but since the vast majority succeeded these problems were ignored.

	C	H	B	E	G	I	T	S	b
C	0.6059	0.0545	0.5142	0.0963	0.1654	0.0181	0.0865	0.0	0.1949
H	0.1003	0.9189	0.0173	0.0010	0.0282	0.0173	0.0322	0.0	0.0003
B	0.0334	0.0016	0.0208	0.0020	0.0074	0.0000	0.0248	0.0	0.0134
E	0.1048	0.0034	0.0706	0.8151	0.0277	0.0193	0.0891	0.0	0.1473
G	0.0355	0.0035	0.0229	0.0035	0.7020	0.0053	0.0141	0.0	0.0011
I	0.0001	0.0000	0.0001	0.0000	0.0000	0.7999	0.0001	0.0	0.0000
T	0.1181	0.0180	0.3535	0.0820	0.0691	0.1401	0.7525	0.0	0.6430
S	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
b	0.0019	0.0000	0.0005	0.0001	0.0002	0.0000	0.0007	0.0	0.0000

Table 3.1: probability matrix

In order to calculate the probability matrix, we want the probability of every possible pair of secondary structure types occurring together. Let $S(c, i)$ denote the secondary structure of chromosome c at residue i . The information we want to find is the conditional probability that $S(c_j, i + 1) = y$ given that $S(c_j, i) = x$. In other words, we want to calculate $P(S(c_j, i + 1) = y | S(c_j, i) = x) \forall x, y \in T$ where T is the set of structure types $T = \{C, H, B, E, G, I, T, S, b\}$. For simplicity, we assume that this probability is independent of the position i in the chain and independent of the type of amino acid at i . Therefore, the probability matrix is defined as $p_{x,y} = P(S(c, i + 1) = y | S(c, i) = x) = n_{xy} / n_x$, where n_{xy} is the number of occurrences of residues with secondary structures x followed by y , and n_x is the total number of pairs that start with x . The n_{xy} and n_x are calculated for every protein structure c_j in our library for $i \in 1, 2, \dots, \text{length}(c_j)$. Our resulting matrix is shown in Table 3.1.

Interestingly, the table is not symmetric. For example, $p_{B,H} = 0.0015$ and $p_{H,B} = 0.04$, which shows that an alpha helix is much more often followed by a beta strand than a beta strand is followed by a helix. The directionality of the chain also manifests itself in the order of secondary structure elements. A possible explanation is that the final secondary structure is influenced by structures formed while the amino acid chain is being constructed one amino acid at a time in a ribosome.

Given the information in the probability matrix, we want to find the crossover

weights for two parents π_1 and π_2 . Let the weights be $W_{\pi_1, \pi_2} = w_1, w_2, \dots, w_n$ where w_i is the weight for choosing crossover point i , for parents π_1 and π_2 . Let the crossover point chosen be j . So if we crossover at point i , (i.e. $j = i$), then in the children the secondary structures at $i + 1$ will be swapped, otherwise they will remain the same as their parents. In other words, if the children results from a crossover at point $j = i$ are c_1 and c_2 , then $S(c_1, i)S(c_1, i + 1) = S(\pi_1, i)S(\pi_2, i + 1)$ and $S(c_2, i)S(c_2, i + 1) = S(\pi_2, i)S(\pi_1, i + 1)$. If the crossover point is at some other point $j \neq i$, then the children contain structures at residues i and $i + 1$ which are the same as their parents, so $S(c_1, i)S(c_1, i + 1) = S(\pi_1, i)S(\pi_1, i + 1)$ and $S(c_2, i)S(c_2, i + 1) = S(\pi_2, i)S(\pi_2, i + 1)$.

The question is, which case ($j = i$ or $j \neq i$) is better? We can use the probability matrix to predict which of the two possible pairs of secondary structures which may be generated by the crossover operator are more likely to occur in real proteins. The probability for case where the crossover point is $j = i$ is:

$$\begin{aligned} & P(\{S(c_1, i + 1) = S(\pi_2, i + 1)\} \wedge \{S(c_2, i + 1) = S(\pi_1, i + 1)\}) \\ & |\{S(c_1, i) = S(\pi_1, i)\} \wedge \{S(c_2, i) = S(\pi_2, i)\}) \\ & = P(S(\pi_2, i + 1)|S(\pi_1, i))P(S(\pi_1, i + 1)|S(\pi_2, i)) \\ & = p_{S(\pi_1, i), S(\pi_2, i+1)} p_{S(\pi_2, i), S(\pi_1, i+1)} \end{aligned}$$

The secondary structures of the two parents and the children are independent of each other, so the probabilities are multiplied. Similarly for case $j \neq i$:

$$\begin{aligned} & P(\{S(c_1, i + 1) = S(\pi_1, i + 1)\} \wedge \{S(c_2, i + 1) = S(\pi_2, i + 1)\}) \\ & |\{S(c_1, i) = S(\pi_1, i)\} \wedge \{S(c_2, i) = S(\pi_2, i)\}) \\ & = P(S(\pi_1, i + 1)|S(\pi_1, i))P(S(\pi_2, i + 1)|S(\pi_2, i)) \\ & = p_{S(\pi_1, i), S(\pi_1, i+1)} p_{S(\pi_2, i), S(\pi_2, i+1)} \end{aligned}$$

To get the weight we take the ratio of the probability of the case $j = i$ to the probability of the case $j \neq i$ and scaling exponent ϵ to get Equation 3.1.

$$w_i = \left(\frac{p_{S(\pi_1, i), S(\pi_2, i+1)} p_{S(\pi_2, i), S(\pi_1, i+1)}}{p_{S(\pi_1, i), S(\pi_1, i+1)} p_{S(\pi_2, i), S(\pi_2, i+1)}} \right)^\epsilon \quad (3.1)$$

The scaling exponent ϵ controls the strength of the weighting. A larger ϵ value causes crossover points with unfavourable structure pairs to be given heavier weights relative to those with favourable structure pairs. The choice of the value for ϵ is explained in Section 4.2.6.

This set of weights is calculated for each pair of parents and used to select a crossover point for them using the roulette selection algorithm as explained in the previous section for operator C1.

3.4.1 Preservation of Tertiary Structure

Methods similar to C1 and C2 could be developed using tertiary structure information instead of, or in addition to secondary structure information.

Tertiary structure identification methods are not as well established. The best option we found was to use a domain identification program which tries to identify the boundaries between domains. There are a number of different programs freely available to do this. One of the most recently developed is called DomainParser[75], which uses network-flow techniques to partition a protein structure into regions with a higher density of interactions, surrounded by lower density of interactions. This will partition a structure into structured domains, with unstructured interconnecting regions in between. Crossover will be biased towards the unstructured interconnecting regions.

To employ this method is outside the scope of this thesis, but in any case would be inappropriate for the set of test proteins we used since they are all very small (10-14 residues) and none of them are large enough to contain multiple domains. We could have tried the method with longer proteins, but Pedersen and Moults software has problems with larger proteins, in that it is prone to crash, and we also found that it runs very slow on larger proteins. For example, for a 46 residue protein called Crambin the MC step was stuck in an infinite loop and unable to complete.

Tertiary structure could be used in an operator similar to C1, where the border points between domains would be given larger weights than those points within domains. The weights determined from tertiary structure information could be combined with the weights from secondary structure information. An operator similar to C2 could also be designed where the weights are based on the probability of different types of domains occurring together. Crossover points would be chosen using the same roulette wheel selection function as for the secondary structure approach.

Chapter 4

Implementation and Experimental Results

This chapter covers the implementation of our new algorithm, the testing procedures, and presents our experimental results.

4.1 Implementation details

Initially we started working on our own GA implementation, based on an implementation done by Stefan Christiansen, Carleton University, Ottawa, ON. His version included a simple one point crossover operator and no mutation operator. It was implemented in Perl to run under Windows and for its energy function it used Swiss PDB Viewer for Windows [31]. It used Swiss PDB viewer's built in scripting language to load PDB files containing protein structures and calculate energy values in order to automate the process. The initial population was generated using FOLDTRAJ [21], a program available as a web service which can generate random conformations consistent with physical constraints and secondary structure predictions.

This program could find structures with a low energy, however it always found structures with *singularities* leading to artificially low energy values for structures

with very low structural similarity to the experimentally determined one. An example where a singularity occurs is when two oppositely charged atoms were nearly overlapping. The energy function contains electrostatic terms $\propto -\frac{1}{x_1-x_2}$ where x_1 and x_2 are positions of two atoms of opposite charge. If the atoms nearly overlap, the magnitude of this term becomes very large, and dominates all of the other terms. So this structure will have a very low energy and the GA will select it. Because of this problem, it was necessary to use a different energy function.

The Amber force field energy for the fitness function was chosen, and the program was ported to the Solaris platform since we had the Amber software suite available on a Sun machine and it would give us access to more computing resources than would be available on a single Windows PC. Initial tests were done using the RMS deviation as the fitness function instead of energy, a method also used by Schulze-Kremer [59]. Unfortunately it could not successfully find conformations with a significant level of structural similarity to the experimentally determined structure, even when just using the RMSD for the fitness function. We added a simple mutation operator which made random changes to the torsion angles at one location at a time which produced a slight improvement but results were still very poor. The problem was attributed to poor quality of the initial population, and the ineffectiveness of the crossover and mutation operators. The very slow computation time for the Amber RMSD function prevented us from doing longer runs of the GA. Also, the operators lacked the capability to restrict structural changes to commonly populated areas of conformational space. This seems to be a necessary feature in most other implementations, which do this by using a library of angles taken from real proteins.

The Amber energy function was tested as the fitness function and it was found to contain singularities. According to [30], the amber force field is not suitable for use with a GA because it is too detailed and computationally expensive, among other reasons.

We also found that all other implementations required a great deal of work on optimization of empirical chemical parameters in order to produce good results, which

requires knowledge of chemistry details outside of our expertise. Also the Perl implementation was not easily parallelizable. For these reasons, we decided to look for an alternative implementation to base our experiments on.

We decided to test our algorithms using the software developed by Pedersen and Moulton [50]. They have published good results produced with their implementation, and they were willing to provide the source code. Their software includes a C program called J which is used to run the GA simulations. The program runs in parallel using PVM and runs on Linux, SGI IRIX and IBM AIX. Modifications were added to their code in order to test the algorithm. About 1500 lines of C++ code were added in order to implement the new crossover operators.

Unfortunately the version of the software provided to us by Jan T. Pedersen is not identical to what Pedersen and Moulton used to obtain the results in their paper since their original version was in Fortran and it was ported to C. The algorithms are the same in the C version except that their crossover operator uses a uniform distribution of crossover points instead of the weighted crossover mentioned in their paper. Also they do not use Monte Carlo local optimization within each generation of the GA since it does not give a significant improvement in performance.

4.2 Experiments

In order to test the program, we used a set of protein sequences with known structure, so that we could compare the structure prediction with the experimentally determined structure (by calculating the RMS deviation), and the sequences were small enough to run simulations in a reasonable amount of time. We chose to use a subset of the sequences used by Pedersen and Moulton in their paper [50]. This allowed confirmation that the program could reproduce their results and that the parameters we used were correct. We also attempted to use the same parameters they did as much as possible. Both modified crossover operators were tested against the base implementation using the same parameters and the same initial populations.

4.2.1 Structure of Experiments

Pedersen and Moult [50] used a set of 28 sequences of independently folding units (IFUs) 10-14 residues in length which were hand-selected from protein sequences with known structure. They applied certain chemical criterion to decide whether a sequence appears to fold independently from the rest of the molecule, so that it should fold correctly on its own in the simulation. Although the simulator can handle longer sequences including whole proteins if they are small enough, we did not use any because of the additional difficulty in setting up the experiment and time constraints.

For each sequence we generated an initial population of structures. The initial population is all based on one topological model generated from the sequence. The bond lengths and many of the bond angles (other than the Φ and Ψ angles and some side chain angles) remain invariant under any of the transformations applied by the MC simulation or the GA.

The initial population is generated by using the topology file as input to the Monte Carlo (MC) simulation. The MC simulation starts by randomizing all of the free bond angles, then it runs for a certain number of steps, sampling the structure at regular intervals. The process continues until enough sample structures are taken for the initial population. The initial population is then used as input to the GA step. The details of the MC and GA steps are described in Section 2.5.2. Their parameters are explained in the following section.

The main output of the experiment is the final population. The lowest energy individual of the final population is taken as the structure prediction. However, since the population always converges so that all individuals have a high structural similarity with each other, the choice is not critical. In addition to the structures, the simulator also has other outputs for checking the performance. After each generation it prints out energy statistics on the population including the maximum and minimum energies, average, standard deviation and median. It prints out diversity which is explained in Section 4.2.3.

It also prints parents of every individual being passed on to the next generation.

The information on the parents of each individual allows the complete genealogy of any individual to be extracted afterwards. Also it keeps a file containing the structure of the lowest energy individual for each generation. These structures compared against the experimentally determined folding and used to generate the RMSD graphs. For some experiments the secondary structures of all individuals in the population were kept in order to examine how secondary structures change over the course of the GA run.

For each test sequence we ran the GA at least 5 times so that we had a large enough sample size to show whether the modified crossover operators perform better than the unmodified version with statistical significance.

4.2.2 Performance Measure

If the modified algorithm provides some improvement, it should be able to produce a structure of better quality in the same number of generations. The measure of the quality of results is the structural similarity (measured with RMS deviation (see Appendix A)) between the conformation with the lowest energy in the final generation of the run, and the experimentally determined structure.

Since the extra computation time for determining secondary structure and applying the modified crossover operator is not significant, it does not need to be taken into account.

The RMS deviation results for different runs varied widely between different runs, therefore to compare performance of different operators, the mean of the RMSD over at least five runs was used to compare the performance of different operators on a particular sequence. The success of the new operators was determined by whether or not the mean RMSD produced using a particular operator was lower than the base version by a statistically significant amount. This comparison was done using t-tests.

4.2.3 Diversity Measurement

Over successive generations in a genetic algorithm the diversity in the population decreases over the course of the run and converges towards an ideal solution. In order to measure diversity, a measure of statistical dispersion of the values in individuals is needed.

In theory the physical entropy of protein structures could be calculated, but according to [59] this problem is not tractable due to the complexity of the interactions between the protein molecule and the surrounding solvent.

In traditional GAs, using a bit string representation, Shannon's entropy measure [62] is often used (e.g. [74]). Equation 4.1 shows how this can be applied to a GA where the bit string chromosome consists of n bits and p_i is the probability that each bit is equal to one and q_i is the probability that each bit is equal to zero.

$$-\sum_{i=1}^n p_i \log p_i - \sum_{i=1}^n q_i \log q_i \quad (4.1)$$

For a GA, usually this equation is applied to the set of chromosomes in the population, which are represented as bit strings of length n . It would be difficult to apply this measure using our representation because the concatenation of all of the floating point numbers used to represent the three dimensional coordinates of each atom would result in a very long bit string, producing a state space many orders of magnitude than the space of the actual problem. It would be possible to obtain the dihedral angles and convert them to a rational number representation which would result in a much smaller bit string.

A simpler method is to use the angular deviation dispersion of the Ψ and Φ angles. Angular deviation is equivalent of standard deviation for angles and it is also a measure of dispersion. In this thesis diversity is measured by taking the angular deviations of the main chain Ψ and Φ angles. The angular deviation of each angle is calculated as follows. Each angle Θ_i is converted to a unit vector using $(x_i, y_i) = (\cos(\Theta_i), \sin(\Theta_i))$. For each angle in the chain, the length of the mean vector is calculated using the formula $r = \sqrt{\bar{x}^2 + \bar{y}^2}$. The angular deviation s (the angular

equivalent of standard deviation) is calculated using $s = 2\sqrt{1-r}$. The mean of the angular deviations over all angles in the chain is used as the diversity measure.

4.2.4 Test Environment

Pedersen and Moulton's program J runs under a number of UNIX variants and has been parallelized using the PVM library. This allowed us to run our experiments on the HPCVL Beowulf cluster at Carleton University. All tests were done using 32 PCs from the cluster which were running Redhat Linux/NPACI rocks. The PCs had 2GHz Intel Xeon processors and 1.5GB of RAM. Each run of the GA took approximately 2-2.5 hours with the parameters given.

4.2.5 Pre-existing Parameters

The three steps of generating the topology, the MC simulation to generate the initial population, and the GA all involve their own parameters. Pedersen and Moulton did experiments to optimize most parameters of the GA as explained in [49] and those experiments were not repeated since we must keep all parameters the same in all experiments in order to correctly compare the different crossover operators. The values were chosen to duplicate the parameters used by Pedersen and Moulton, except where noted. This section explains their parameters. The next section explains new parameters for C1 and C2 and the experiments done to optimize them.

The topology generation step involves a set of covalent geometry parameters which determine the bond lengths and bond angles used to construct the topology. There are a number of different geometries in use which have slightly different parameters. Pedersen and Moulton primarily used the covalent geometry from a molecular dynamics package called GROMOS, although they performed tests using several others and they noted that the quality of the results was extremely sensitive to the geometry used [50]. This is because their energy function was tuned using a particular geometry and using any other geometry seems to cause an imbalance between terms in the energy

function. We also noted this problem, since initial tests were done using topology files generated with Amber. Nearly all of the sequences folded into alpha helices, but these results were discarded since we could obtain much better results when using Pedersen and Moult's GROMOS topology files.

The MC simulation and the GA both share the same parameters for the energy function. The energy function contains a number of different terms for calculating the contributions of various forces involved. The terms used in the energy function were: clash, local, phobe, solvent, hbond, and sidechain.

Where:

- clash - clash energy, a rejection term if there is an overlap larger than 0.5 Å between two atoms or whatever the Overlap parameter is set to.
- local - local main chain electrostatics
- phobe - hydrophobic potential
- solvent - area dependent solvation term
- hbond - non-local main chain electrostatics
- sidechain - side chain - side chain electrostatics

Where 'local' corresponds to G_{local} in the paper. G_{ES} (in paper) corresponds to $hbond + sidechain$. $G_{solv} = phobe + solvent$ [47]. They also have the parameter Overlap used by the clash component of the energy function which sets the maximum overlap of van der Waals spheres of atoms without penalty. Overlap was set to 0.35Å.

The MC simulation also has the parameters *NSteps*, which sets the total number of steps and *Monitor*, which sets the length of interval to take samples for the initial population. We used $NSteps = 20000$ steps and set $Monitor = 100$ to generate an initial population of 200, since Pedersen and Moult found these values to be adequate. In the reproduction step the population is doubled to get a running population of 400. There is also a number of parameters to control the proportions of different types of

moves made on the molecule (explained in Section 2.5.2). The mixture of moves used was: 50% backbone and 50% sidechain, and the types of moves were 50% library, 25% relative and 25% Shoshona. The parameter `MoveAngle` (set to 5.0) controls the size of local moves. The parameters `InitialTemperature` and `FinalTemperature` control the temperature used in the application of the Metropolis rule. In the paper, the `InitialTemperature` was 300.0K (about room temperature) and the `FinalTemperature` was 174.0K. However, using a lower final temperature only seems to cause premature convergence, so it was changed to 300.0K for a constant temperature simulation.

In the GA the population size was 400, and the simulation was run for 60 generations in most cases since this was sufficient for convergence both in Pedersen and Moults paper and in our own tests. In their tests they ran for 100 generations, but using the minimum number of generations necessary for convergence reduced the time required to run the experiments. Elitism is controlled with the parameter `CarryOverFraction` (set to 0.1). This means that the top 10% of the population is carried over to the next generation unmodified. `NPick` controls the number of the joint angle trials and was set to 50 since Pedersen and Moults found 50 to be sufficient. During a crossover operation, this number of structures, each with different randomly selected dihedral angles at the crossover point, are evaluated and the lowest energy individual is selected for the child. The parameter `GAMcSteps` controls the number of steps of MC minimization to run at each generation but this was set to 0 for all our tests since it only seems to decrease the quality of the results, and it is time consuming. This eliminated the Lamarkian component of the GA.

4.2.6 C1 and C2 Parameters

In choosing the parameters for C1 and C2, which control the strength of crossover point weights, the goal was to find parameters to give improvements in performance for as many sequences as possible, using the results obtained using different parameters values. The amount of data was available was limited by constraints on time and computing resources.

Operator C1 has the parameters `BORDER_WEIGHT`, and `COIL_WEIGHT`. These parameters set the relative weights of border points between different structure types and the weight of unstructured coil regions relative to the default weighting of 1.0. It is expected that setting the weights too high would cause premature convergence or stagnation, and setting them too low would give no difference in performance from the unmodified crossover operator in the case where `BORDER_WEIGHT` = `COIL_WEIGHT` = 1. In Figure 4.1 the structural similarity of the product of the simulation to the correct structure is shown for different values of `BORDER_WEIGHT` and `COIL_WEIGHT` for protein sequence 1mbc 131-142. Since there is a local (if not global) minimum in the RMSD when either parameter is set to 4, both parameters were set to 4, the remaining tests on operator C1 presented in this thesis were done with `BORDER_WEIGHT`=`COIL_WEIGHT`=4 unless noted otherwise. Other tests were done using `BORDER_WEIGHT`=1.5, `COIL_WEIGHT`=1.25, but these produced an overall average improvement in RMSD over the normal operator of only 6%, indicating that these weights are too weak.

In Figure 4.2 the diversity for different sets of parameters is compared. The two lowest diversity curves (bw6 and cw10) are for higher weight values, indicating that the higher weights caused faster convergence on average, however the structural similarity results were not better, therefore these higher weights caused premature convergence in some cases.

Operator C2 has the parameter ϵ , which controls how strong the weights are. As with C1, using a value too large would cause premature convergence or stagnation, or a value too small would cause the weightings to be too weak to have any effect. In Figure 4.3 the structural similarity results for different values of ϵ are shown for protein sequences 1mbc 131-142 and 1bgs 88-98. For both sequences there is a local minimum near $\epsilon = 0.5$ but for 1bgs 88-98 results are better when ϵ is 3 or 4. For sequence 1mbc 131-142 there is only a marginal improvement when $\epsilon > 2$. Other tests were done on all of the test sequences using $\epsilon = 0.5$. In those tests some of the sequences performed better and some worse but the overall improvement was about the same (18% instead of 16%). Unfortunately there does not appear to be

Figure 4.1: Structural similarity results vs. C1 parameter values. For each series one of the parameters is kept constant at one while the other is varied and the result when both parameters are set to 4 is also shown for comparison.

Figure 4.2: Diversity at each generation using operator C1. Each series gives the diversity at each generation averaged over five runs for operator C1 on sequence 1mbc 131-142. In the legend bw indicates that BORDER_WEIGHT is set to the value given and COIL_WEIGHT is set to 1, and cw indicates the opposite.

Figure 4.3: Structural similarity results averaged over five runs for different values of ϵ with C2

a setting which gives the best performance for all sequences. The parameter setting $\epsilon = 2$ was chosen because it performs better on sequence 1bgs 88-98 which is the only one we tested that does not have an alpha-helix structure, and it should work better on general sequences. This setting was used for the remaining tests of operator C2 on other sequences which are presented here except where specified other values are specified.

In Figure 4.5 the diversity for runs with C2 with several values of ϵ for sequence 1bgs 88-98 are shown. For increasing values of epsilon the diversity curves become less smooth. This suggests that the higher values of ϵ cause the GA to get caught in local minima where the GA stagnates for a while and increases the diversity until it escapes the local minima. This might reduce the efficiency of the GA if the effect is too strong, and may account for the reason why performance only improves slightly

Figure 4.4: Diversity at each generation using operator C2 on 1bgs 88-98. Each series gives the diversity averaged over five runs.

when $\epsilon > 1$ for this sequence.

The diversity results for C2 with various values of ϵ for sequence 1mbc 131-142 are in Figure 4.5. For this sequence the higher values of epsilon cause the diversity to decrease more steeply in earlier generations but remain higher in later generations. For this sequence $\epsilon = 4$ the structural similarity results still continue to improve slightly and the point where increasing epsilon further hinders performance was not found. The faster initial decrease in diversity suggests that higher weights guide the search to a structure in the vicinity of the correct one faster, and then in later generations the diversity remains higher, so searching still continues in that vicinity, encouraging it to find lower energy structures with higher structural similarity to the experimentally determined structure.

Figure 4.5: Diversity at each generation using operator C2 on 1mbc 131-142. Each series gives the diversity averaged over five runs.

protein	Range	Energy	Correct secondary structure
1bgs	88-98	-23.9	-EE-TT--EE-
1hrc	92-103	-26.7	-HHHHHHHHHH-
1lmb	15-26	-24.4	-HHHHHHHHHH-
1mbc	6-17	-24.1	-HHHHHHHHHH-
1mbc	131-142	-25.4	-HHHHHHHHHH-
5cyt	88-101	-30.7	-HHHHHHHHHH-

Table 4.1: Protein sequences used for testing. The columns show the name of the protein, the subsequence of the protein used, the energy of the experimentally determined structure using the energy function in Pedersen and Moults program, and the correct secondary structure.

4.2.7 Test sequences

The protein sequences we used for testing are shown in Table 4.1. We chose sequences that contain primarily alpha helices in their experimentally structures from the test set used in Pedersen and Moults paper. This was because we had problems reproducing results as good as those reported in [50], or as good as the test results included with the source code of their program (which are all either the same or slightly better than in the paper). However, we were able to produce alpha helical structures nearly as good as theirs. We included one structure which does not contain alpha helices (1bgs 88-98). We used their software without modifications and attempted to duplicate the parameters they used as closely as possible. We also tried changing a number of parameters to see if it would improve the results including: GA parameters GAMCSteps, Generations, and NPick; MC parameters NSteps and NSteps; and some parameters common to both including Overlap, and FinalTemperature. But we were unable to obtain significantly better results by modifying any of these parameters.

The RMSD results for some of the sequences using the unmodified version of the software are shown in Table 4.2. Some other sequences were tested but the results are omitted because they were done using the Amber covalent geometry (which generally produces lower quality results) instead of GROMOS. The program could not achieve as low RMSD values as those reported in that paper for most of the sequences, although the structures containing mainly alpha helices worked better.

Sequence	Our RMSD	n	Pedersen & Moul't's RMSD
1bgs 88-98	3.112	5	2.705
1hrc 92-103	1.285	9	0.421
1mbc 131-142	1.230	8	0.214
1mbc 6-17	1.606	8	0.353
1lmb 15-26	0.876	7	0.428
5cyt 88-101	1.7125	5	1.22
1fkf 27-38	4.878	1	2.44
1mbc 29-40	2.635	1	1.14
1alc 21-32	2.475	1	2.07

Table 4.2: comparison between Pedersen & Moul't's RMSD (from one run) and ours (averaged over the number of runs in column "n") using an unmodified version of their software and parameters matched to theirs to the best of our knowledge.

In the paper they observed that their simulation could fold helices more easily than other types of structures. Other types of secondary structures are more sensitive to small variations in the parameters and the covalent geometry model used to generate the initial population.

We observed this problem when initially we used Amber covalent geometry to construct the topologies of the molecules, whereas Pedersen and Moul't used the GROMOS model. These models determine the bond lengths and angles which do not change through the course of the model run. Using the Amber topology files the simulator could only produce alpha helix secondary structures, but using Pedersen and Moul't's topology files allowed it to produce other types of secondary structures and results with lower RMSD to the experimentally determined structures. However, the simulator still does not perform as well as it does for their results. There are two possible explanations for this discrepancy. Our parameters for the GA or MC steps may have been slightly different from theirs, despite our best efforts to duplicate them using information obtained in the paper and by communication with Jan Pedersen. The other possibility that the C port provided to is slightly different from the original Fortran implementation.

Figure 4.6: The entire initial population (on the left) and the final population (on the right) fitted to the experimentally determined structure for sequence 1hrc 92-103 from a representative run of the GA.

4.3 Experimental Results

4.3.1 Overview

In this chapter we present the experimental results for 6 protein sequences using our program. The energy and RMSD of structures produced by the normal (unmodified) version are compared with results from operator C1 and C2.

Figure 4.6 shows all of the structures in the initial population and in the final population fitted to the experimentally determined structure for a representative run of the GA. In the initial population the structures are very diverse, while in the final population it has converged close to the experimentally determined structure. This example shows that the GA is working properly, and it is representative of the behaviour of the other test sequences as well. A graph of diversity for the same sequence is shown in Figure 4.7, which also confirms that the structures converge towards the end of the run. Interestingly, diversity actually increases at the beginning of the run. This is because the crossover operator introduces the diversity by changing the torsion angles at the crossover point, and at the beginning of the run the selection pressure is not strong enough to offset this increase in diversity.

In Figure 4.8 the best structural similarity (lowest RMSD) results produced by

Figure 4.7: The diversity measured on a representative run of the GA on sequence 1mbc 131-142

the program out of several runs using each of the three operators are fitted to the experimentally determined structures. For the cases where operator C1 or C2 produced a lower RMSD result, the structures can also be visually observed to be closer to the experimentally determined structure.

Figure 4.9 and Table 4.3 summarize the RMSD results for 6 protein sequences compared to their experimentally determined structures for the 3 crossover operators. In 5 of the 6 cases, operator C1 gives a lower final RMSD result than the unmodified version and an overall average of 16% better results. Operator C2 also gives a lower final RMSD result in 5 out of 6 cases, and an overall average of 18% better results.

4.3.2 Analysis of Results

Figures B.1 through B.11 in Appendix B are graphs of the RMSD between lowest energy individual in each generation and the experimentally determined folding, and the energy of the lowest energy individual in each generation. Each data series represents a complete run of the GA. Every run is different because of the non-deterministic

Figure 4.8: The lowest RMSD result produced by each operator for each sequence fitted to the crystal structure. Black, blue, red and green correspond to experimental, normal, C1 and C2 respectively.

Figure 4.9: Summary of our RMSD results for the normal (unmodified) crossover operator, C1 and C2. The RMSD values are averaged over the number of runs indicated in Table 4.3.

Sequence	normal	C1		C2	
	mean±stdev	mean±stdev	%imp.	mean±stdev	%imp.
1bgs_88-98	3.06±1.36	1.93±1.48	36.8%	2.39±0.77	22.0%
1hrc_92-103	1.28±0.15	1.19±0.14	7.2%	1.35±0.23	-5.2%
1lmb_15-26	0.88±0.20	1.10±0.15	-25.9%	0.86±0.33	2.1%
1mbc_131-142	1.21±1.05	0.60±0.20	50.3%	0.76±0.33	37.1%
1mbc_6-17	1.61±1.12	1.49±1.76	7.2%	1.05±0.23	34.4%
5cyt_88-101	1.71±0.98	1.57±1.35	8.6%	1.60±0.75	6.8%
Average improvement over normal			14.0%		16.2%

Table 4.3: Summary of RMSD results for six protein sequences comparing our results for operators C1 and C2 against our results for the normal (unmodified) crossover operator. The RMSD results (in Å) are averaged over at least 5 runs of the GA and the standard deviations of the results are also included. The “%imp.” columns show the percentage of improvement of C1 and C2 compared to the normal version. The average improvements are shown in the last row.

nature of a GA. From one generation to the next, the energy never increases because elitism ensures that the lowest energy individual is always carried over to the next generation though it may remain constant for several generations, especially as the population begins to converge. The RMSD of the lowest energy individual may increase from one generation to the next (i.e. its structural similarity to the experimentally determined structure decreases) even though the energy has decreased. There is no reason why the molecule's RMSD should always decrease as it folds, however increases sometimes may indicate problems with the energy function.

For all cases, in our results and in Pedersen and Moul't's results, the program always produces structures with energy values much lower than the energy of the correct structure (as we can see from comparing the energies of the experimentally determined structures measured with Pedersen and Moul't's energy function in Table 4.1 with the energy values displayed in Figures B.1 through B.11). The reason for this discrepancy could be because either the correct structure is not the minimum energy structure, or because the IFU is not completely independent, but it is mostly likely because of problems in the energy function. In general, the quality of the energy function is believed to be the main limiting factor for protein folding with GAs [61] and there does not seem to be a consensus among different researchers as to what energy function produces the best results. Despite this problem, the program produces structures with low RMSD when compared to their correct structures.

The performance of the different crossover operators varies significantly for different protein sequences. For 1bgs 88-98 in Figures B.1, B.2 and B.3, the RMSD does not significantly improve after 30 generations even though the energy continues to decrease. Run C1_2 is a particularly bad case, where the RMSD remains high through the latter part of the run, even though its final energy value is about average among those runs. It converged to a local minima which has approximately the same energy as the other results, but it is structurally distant. This particular sequence probably has a large number of local minima with about the same energy, but they are all significantly structurally different. At the 20th generation of the best runs of each operator there is a difference in the secondary structures of the lowest energy individuals.

The experimentally determined structure for this sequence has secondary structure CEECTTCCEEC, with three secondary structure elements, two residues long each. At the 20th generation the secondary structures of the lowest energy individuals for the normal operator, C1 and C2 are CTTBTTTBTTT, CEEETTTEEEEC, and CEEETTTEEEEC respectively. The individuals from C1 and C2 have the same secondary structure, with 8 of 11 residues correct, but for the normal operator only has 3 correct. This strongly suggests that C1 and C2 increased the performance by encouraging the formation of correct secondary structures. C1 performs better because C2 strongly favours longer secondary structure elements, but this sequence only contains very short ones.

For 1hrc 92-103 (Figures B.4, B.5 and B.6) C1 and C2 produced lower energy structures on average, but for C1 the average RMSD is only slightly lower, and for C2 the average is higher. The RMSD results were very slightly better for C2 with $\epsilon = 0.5$. These results suggest that C1 and C2 are more successful than the normal operator with searching the energy function, but the energy function is not accurate enough to find a better structure.

For 1lmb 15-26 (Figures B.13, B.14 and B.15), as with 1hrc 92-103, both C1 and C2 produced significantly lower energy individuals than the unmodified crossover, but in terms of structural similarity to the experimental structure, neither one performed well. This suggests that there is a problem with the accuracy of the energy function, since individuals with lower energy are further from the correct structure. C2 performed worse than the normal operator on average. Figure 4.10 shows that in the runs using C2 the diversity decreased more quickly during generations 9-15, which suggests premature convergence, and may explain why C2 performs more poorly than C1.

For 1mbc 6-17 (Figures B.7, B.8 and B.9), the normal operator and C1 had at least one run which produced a high RMSD early on and kept it until the last generation. C2 does not have any runs which failed in this way which may only be coincidental. Those runs were: normal 2, 5 and 6, and C1.3. These runs are also the ones which had the highest energies. In those cases the energy function correctly assigned a high energy to the unfit individuals, however the search procedure failed to find a lower

Figure 4.10: Diversity averaged over all runs with operator C1, and over all runs with operator C2.

energy individual. The population seems to have converged in a local minima too early on, and it is likely that operator C1 encouraged this to happen by narrowing the choice of crossover points too much. Interestingly, all of the poor runs mentioned above contained an alpha helix from residues 6 to 9 and sometimes including residues 5 and 11, and the other residues were coil (unstructured), whereas all of the low RMSD results contain a longer alpha helix from residues 2 to 10. Operator C2 probably performed better because it favours long alpha helices, grouping pairs of residues which are in an alpha helix structure together rather than grouping an alpha helix next to a coil. C2 favours long alpha helices because, according to the probability matrix, the probability of finding a residue with helical structure followed by another is $p_{H,H} = 0.9189$, but the probabilities of finding a residue in coil structure followed by a residue in helix structure or vice versa are only $p_{C,H} = 0.0545$ and $p_{H,C} = 0.1003$ respectively.

For 5cyt 98-101 (Figures B.16, B.17, and B.18), all of the runs converged to approximately the same energies for all operators. The difference in the average RMSD of the different operators is mainly because some of the runs converged to structures with a high RMSD between 2.5 to 4Å while most of them converged to about 1Å. This indicates that there are other local minima in the energy surface with similar energies to the correct structure's energy, but they have low structural similarity to the correct structure, and in some cases the energy function fails to distinguish between them. The runs with a high final RMSD (normal_3, normal_4, C1_4, C2_1, and C2_2) only have helical structure in residues 5-7, whereas the successful runs nearly all have an alpha helix spanning the entire chain. Run C2_2, which was produced with operator C2, has the longest alpha helix with 7 residues. Operator C2's high RMSD runs are not as poor, which suggests that while the energy function failed to select better structures, the crossover operator strongly favoured long alpha helices improving the result.

In many of the runs (normal_3, normal_4, C1_3, C1_4, C2_1, C2_2) the lowest energy structure changes back and forth between low RMSD and high RMSD structures in successive generations. This indicates that there are two or more competing

families of structures in different local minima and the one which produces the lowest energy structure changes several times. For example, C_1_3 flips to a high RMSD in generation 19 and goes back down at generation 26. From generations 19 to 25 the lowest energy structure has secondary structure CGGGTTHHHHHHC, then in generation 26 it changes to CCHHHHHHHHHHC which is only has one residue with incorrect secondary structure. These two secondary structures seem to characterize the competing families. In generations 19-25 there were many individuals with secondary structure CCHHHHHHHHHHC, but they had higher energies, but at the 26th generation the genealogy information shows that two individuals with this same secondary structure were combined by the crossover operator to produce a highly fit individual with the lowest energy. The other runs which converged to a high RMSD also had incorrect secondary structures. Operator C2 produced much better results for this sequence when $\epsilon = 0.5$ (RMSD=1.156, an improvement of 32.5% over the normal operator) which indicates that $\epsilon = 2$ is too strong a weighting for this sequence. Although unfortunately diversity data is not available for comparison, this suggests that the stronger weights produced by the higher ϵ value cause stagnation or premature convergence, which is more likely with this sequence because it contains structurally distant local minima.

For 1mbc 131-142 (Figures B.10, B.11, and B.12) both C1 and C2 performed well. They produced structures with lower energies on average as well as lower RMSD. This indicates that they were more successful in searching for lower energy structures, and that the lower energy structures actually had lower RMSD values as well. This is in contrast to the results for 1lmb 15-26 and 1hrc 92-101 where the structures found with lower energy values had higher RMSD values due to inaccuracies in the energy function. In run normal.3 the lowest energy individual produced has relatively high RMSD and energy, indicating a failure in the search function. For this sequence C2 performed better when $\epsilon = 0.5$ giving an improvement of (47.2%) over the normal operator, therefore the setting $\epsilon = 2$ weights too strongly for this sequence.

The improvements of C1 and C2 over the normal operator unfortunately are not statistically significant. The best result from Table 4.3, where C1 gave an average improvement of 50% over the normal operator for sequence 1mbc 131-142 was analyzed

with a t-test. T-tests can be used to show whether the difference between the means of two groups is significant for a small sample size. Using a one tailed t-test assuming unequal variances, and assuming the variables have normal distributions, the t-test says the probability that there is no difference between the mean RMSD values for normal and C1 on sequence 1mbc 131-142 is 0.057. Unfortunately this is just greater than 0.05. If it had been less than 0.05 we would be able to say that the mean RMSD for C1 is lower than the normal operator with 95% confidence. Therefore the result is not significant. An additional 5 runs were done for this test case but significance was still not shown. For the other top 3 best improvements t-tests were also done but significance was not shown.

A box plot for the results of the normal operator and C1 on sequence 1mbc 131-142 is shown in Figure 4.11. Intuitively, in the plot one can see that it is highly likely that the normal population could contain the points for C1, which explains why the t-test does not show that these two populations are different. As shown in Table 4.3, most of the runs have a high standard deviation in results, and this prevents the t-test from showing that the groups are different. The high standard deviations in results are probably related to the same problem which prevented us from duplicating Pedersen and Moulton's results, since [49] does not mention that the structural similarity of their results varies significantly between runs.

In general, operators C1 and C2 perform differently on different protein sequences because the the landscape of the energy function's hyper-surface, as well as its accuracy vary greatly between sequences. Additionally, modifications to the parameters in C1 and C2 have different effects on different sequences. Some sequences have many significantly different structures with approximately the same energy (e.g. 1hrc 98-103, 1lmb 15-26) and the energy function fails to discriminate among them, while for others the lowest energy individual is always the one with the lowest RMSD and performance is limited by searching efficiency. Due to problems with our implementation beyond our control we were unable to show that the operators give an improvement with a high level of statistical significance. But in general, even though the energy function accuracy varies, operators C1 and C2 seem to improve the efficiency of the

Figure 4.11: A box plot comparing 1mbc 131-142 for normal operator (group A) and C1 (group B). The Y axis is RMSD. The boxes contain the middle 50% of the population, the line in the middle of the box is the median and the markers above and below the boxes represent the maximum and minimum values.

searching of the GA, except in a few cases where they caused premature convergence.

Chapter 5

Conclusion

In this thesis we have presented two crossover operators for application to the protein folding problem and experimental results. These operators use secondary structure information from the running population to weight the choice of crossover points in order to preserve secondary structure elements. They were implemented and tested with a pre-existing protein folding simulation program on a number of sequences and produced structures with better structural similarity with the experimentally determined structure compared to the basic crossover operator in 5 out of 6 cases for both C1 and C2. On average operator C1 produced structures 14% better than the normal crossover operator and operator C2 produced 16% better, but the statistical significance of these results was not proven. The hypothesis that using secondary structure information to choose crossover points can improve the performance of a GA with protein folding simulation has been supported.

5.1 Future work

In order to use these operators for more practical applications, they would have to be tested on larger protein sequences and on more types of structures. We did not attempt to fold full protein sequences since the software is prone to crash for

longer sequences. Also we did not have time to run the experiments since the time the simulator takes to run increases exponentially with the length of the sequence. However, these operators were designed with scaling to larger proteins in mind, and we hope that they would give larger improvements on longer sequences with more secondary structure elements.

Also for longer protein sequences it would be possible to use tertiary structure information to weight crossover points in order to preserve domains. Another possible approach, inspired by domain partitioning methods, is to form a graph of interactions between atoms where the edges represent the favourability of the interaction, and crossover points would be weighted towards cutting the maximum number of unfavourable interactions.

Operator C2 could be improved so that, instead of being using only patterns of pairs, by using patterns of fragments of 3 or more residues. Some GA implementations have used libraries of fragments for constructing the initial population and performing mutations (e.g. [65]).

Another idea is to use secondary structure predictions in creating the initial population and use the crossover weights to help preserve it. Most secondary structure prediction programs also assign a confidence level along with the prediction for each residue. So the initial population could be generated by assigning predicted secondary structures to particular residues proportional to their confidence values, the other portion being randomly selected structures, perhaps by MC simulation. Then the crossover operator would assign crossover weights to particular points in the sequence proportional to the confidence value, since lower confidence values mean that it would have to do a more extensive search in this region in order to find the correct structure. Also coil regions of the chain would be given higher weights since they are more variable than the structured regions. The weights would remain constant throughout the GA run or perhaps be decreased as the secondary structures become fully formed.

Operator C1 and C2 could also be used together. This could be done by randomly

choosing between applying operator C1 or C2 for each crossover operation. The ratio of applying C1 and C2 could be tuned manually by experimentation or a self-adaptation procedure similar to [19] could be used. The ratio would be adapted in real time through the course of the run, based on evaluating the relative fitness of the offspring produced by them. This approach is appropriate since the effectiveness of the operators varies greatly for different protein sequences.

Appendix A

Definition of Root Mean Square Deviation

The root-mean-square deviation (RMSD) is used in this thesis to measure the structural similarity between two protein different structures with the same amino acid sequence and chemical composition. In particular, we used it to compare protein structures produced by simulation with the experimentally determined structure.

It is defined in Equation A.1 for N atoms where u_i and v_i are three dimensional coordinates of the corresponding atoms in the two structures. Usually only the coordinates of the C_α atom from each of the amino acids is used. Throughout this thesis when we mention the RMS deviation or RMSD we mean the RMS deviation of the C_α atoms only.

$$\sqrt{\sum_i^N (u_i - v_i)^2} \quad (\text{A.1})$$

The RMS deviation values are measured in Angstroms ($\text{\AA}$), where $1\text{\AA} = 10^{-10}$ metres. An RMSD of 0 indicates that the structures are identical. To give a feel for the magnitudes of these values, for small protein chains of 10–50 residues, 0 to $3\text{\AA}$ indicates strong structural similarity which can be visually confirmed; 3 to

6Å indicates weak similarity, and larger values indicate very weak, if any similarity. The RMS value is dependent on the number of atoms used. Although there is size independent method for comparing structural similarities [42], which avoids problems when comparing the deviations of different sized proteins, it was not necessary to use it because all of our test files are about the same size.

For the results reported in this thesis the program Carnal from the Amber 6 suite was used to calculate RMSD. Amber is a set of tools for molecular dynamics simulations on biomolecules.

Appendix B

RMSD and Energy results

Figure B.1: 1bgs 88-98 normal operator

Figure B.2: 1bgs 88-98 operator c1

Figure B.3: 1bgs 88-98 operator c2

Figure B.4: 1hrc 91-103 normal operator

Figure B.5: 1hrc 92-103 operator c1

Figure B.6: 1hrc 92-103 operator c2

Figure B.7: 1mbc 6-17 normal operator

Figure B.8: 1mbc 6-17 operator c1

Figure B.9: 1mbc 6-17 operator c2

Figure B.10: 1mbc 131-142 normal operator

Figure B.11: 1mbc 131-142 operator c1

Figure B.12: 1mbc 131-142 operator c2

Figure B.13: 1lmb 15-26 normal operator

Figure B.14: 1lmb 15-26 operator c1

Figure B.15: 1lmb 15-26 operator c2

Figure B.16: 5cyt 88-101 normal operator

Figure B.17: 5cyt 88-101 operator c1

Figure B.18: 5cyt 88-101 operator c2

Bibliography

- [1] I. P. Androulakis, C. D. Maranas, and C. A. Floudas. Global minimum potential energy conformations of oligopeptides. *J. Global Optim.*, 11:337–363, 1997.
- [2] C. Anfinsen. Principles that govern the folding of protein chains. *Science*, 181:223–230, 1973.
- [3] Bonnie Berger and Tom Leighton. Protein folding in the hydrophobic-hydrophilic (HP) is NP-complete. In *Proceedings of the second annual international conference on Computational molecular biology*, pages 30–39. ACM Press, 1998.
- [4] Eckart Bindewald, Jurgen Hesser, and Reinhard Manner. Implementing genetic algorithms with sterical constraints for protein structure prediction. In *LNCS – PPSN III*, volume 1498, pages 959–967. Springer-Verlag, 1998.
- [5] Philip E. Bourne and Helge Weissig, editors. *Structural Bioinformatics*, chapter 25, pages 509–524. Wiley-Liss, 2003.
- [6] Philip E. Bourne and Helge Weissig, editors. *Structural Bioinformatics*, pages 499–500. Wiley-Liss, 2003.
- [7] Philip E. Bourne and Helge Weissig, editors. *Structural Bioinformatics*, chapter 27. Ab Initio Methods, pages 547–557. Wiley-Liss, 2003.
- [8] Philip E. Bourne and Helge Weissig, editors. *Structural Bioinformatics*, chapter 18, page 341. Wiley-Liss, 2003.

- [9] J. U. Bowie and D. Eisenberg. An evolutionary approach to folding small α -helical protein that uses sequence information and an empirical guiding fitness function. *Proceedings of the National Academy of Science USA*, 91, 1994.
- [10] H. Chan and K. Dill. The protein folding problem. *Physics Today*, 46(2):24–32, 1993.
- [11] James Cheetham. Protein architecture class lecture slides. Carleton University, 2003.
- [12] C. Colovos and T. O. Yeates. Verification of protein structures: Patterns of nonbonded atomic interactions. *Protein Science*, 2:1511–1519, 1993.
- [13] T. H. Cormen, C. L. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 15.3 Elements of dynamic programming. MIT Press, Cambridge, MA, 2nd edition, 2001.
- [14] David Corne and Gary B. Fogel. *Evolutionary computation in bioinformatics*. Morgan Kaufmann, Amsterdam ; New York, 2003.
- [15] Thomas E. Creighton. Protein folding. *J. Biochemistry*, 270:1–16, 1990.
- [16] Pierluigi Crescenzi, Deborah Goldman, Christos Papadimitriou, Antonio Piccolboni, and Mihalis Yannakakis. On the complexity of protein folding (extended abstract). In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 597–603. ACM Press, 1998.
- [17] T. Dandekar and P. Argos. Folding the main-chain of small proteins with the genetic algorithm. *Journal of Molecular Biology*, 236:844–861, 1994.
- [18] T. Dandekar and P. Argos. Identifying the tertiary fold of small proteins with different topologies from sequence and secondary structure using the genetic algorithm and extended criteria specific for strand regions. *Journal of Molecular Biology*, 256:645–660, 1996.

- [19] L. D. Davis. Adapting operator probabilities in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 61–69. Morgan Kaufmann, 1989.
- [20] Volker A. Eyrich, Daron M. Standley, Anthony K. Felts, and Richard A. Friesner. Protein tertiary structure prediction using a branch and bound algorithm. *PROTEINS: Structure, Function, and Genetics*, 35:41–57, 1999.
- [21] H.J. Feldman and C.W.V. Hogue. A fast method to sample real protein conformational space. *Proteins*, 39:112–131, 2000.
- [22] C. A. Floudas. Global optimization approaches in protein folding and peptide docking. In *Proceedings of DYCOPS-5 on Dynamics and Control of Process Systems*, pages 167–178, 1998.
- [23] S. Forrest and M. Mitchell. Relative building-block fitness and the building block hypothesis. In L. D. Whitley, editor, *Foundations of Genetic Algorithms 2*. Morgan Kaufmann, San Mateo, Calif, 1993.
- [24] D. Frishman and P. Argos. Knowledge-based secondary structure assignment. *Proteins: structure, function and genetics*, 23:566–579, 1995.
- [25] K. D. Gibson and H. A. Scheraga. Revised algorithms for the build-up procedure for predicting protein conformations by energy minimization. *Journal of Computational Chemistry*, 8:826–834, 1987.
- [26] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, San Mateo, Calif., 1991. Morgan Kaufmann.
- [27] D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and first results. *Complex Systems*, 3:493–530, 1989.
- [28] Sridhar Govindarajan and Richard A. Goldstein. On the thermodynamic hypothesis of protein folding. *Biophysics*, 95:5545–5549, 1998.

- [29] G. W. Greenwood. Revisiting the complexity of finding globally minimum energy configurations in atomic clusters. *Zeitschr. für Physik. Chem*, 211:105–114, 1999.
- [30] Garrison W. Greenwood and Jae-Min Shin. On the evolutionary search for solutions to the protein folding problem. In David Corne and Gary B. Fogel, editors, *Evolutionary computation in bioinformatics*. Morgan Kaufmann, Amsterdam ; New York, 2003.
- [31] N. Guex and M.C. Peitsch. Swiss-model and the swiss-pdbviewer: An environment for comparative protein modeling. *Electrophoresis*, 18:2714–2723, 1997.
- [32] J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.
- [33] Tianzi Jiang, Qinghua Cui Guihua Shi, and Songde Ma. Protein folding simulations of the hydrophobic–hydrophilic model by combining tabu search with genetic algorithms. *Journal of Chemical Physics*, 119(8), 2003.
- [34] D.T. Jones. Protein secondary structure prediction based on position-specific scoring matrices. *J Mol Biol*, 292:195–202, 1999.
- [35] Wolfgang Kabsch and Christian Sander. Dictionary of protein secondary structure: pattern recognition of hydrogen-bonded and geometrical features. *Biopolymers*, 12:2577–2637, 1983.
- [36] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, Number 4598, 13 May 1983*, 220, 4598:671–680, 1983.
- [37] Rainer König and Thomas Dandekar. Improving genetic algorithms for protein folding simulations by systematic crossover. *BioSystems*, 50:17–25, 1999.
- [38] John Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. The MIT Press, 1992.
- [39] Natalio Krasnogor, William E. Hart, Jim Smith, and David A. Pelta. Protein structure prediction with evolutionary algorithms. In Wolfgang Banzhaf, Jason

- Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1596–1601, Orlando, Florida, USA, 1999. Morgan Kaufmann.
- [40] Gary B. Lamont and Laurence D. Merkle. Toward effective polypeptide structure prediction with parallel fast messy genetic algorithms. In *Evolutionary computation in bioinformatics*, pages 138–161. Morgan Kaufmann, Amsterdam ; New York, 2003.
- [41] R. Luthy, J. Bowie, and D. Eisenberg. Assessment of protein models with three-dimensional profiles. *Nature*, 356:83–85, 1992.
- [42] V. N. Maiorov and G. M. Crippen. Size independent comparison of protein three-dimensional structures. *Proteins: Structure, Function and Genetics*, 22:273–283, 1995.
- [43] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087, 1953.
- [44] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996.
- [45] R. Pachter, S. S. Patnaik, R. L. Crane, and W. W. Adams. New smart materials: molecular simulation of nonlinear optical chromophore-containing polypeptides and liquid crystalline siloxanes. *SPIE Proc.*, 1916:2–13, 1993.
- [46] Arnold L. Patton, W. F. Punch III, and E. D. Goodman. A standard GA approach to native protein conformation prediction. *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 574–581, 1995.
- [47] Jan Pedersen. Personal communication, Dec 2003.
- [48] Jan T. Pedersen and John Moult. Genetic algorithms for protein structure prediction. *Current Opinion in Structural Biology*, 6:227–231, 1996.

- [49] Jan T. Pedersen and John Moult. Ab initio protein folding simulations with genetic algorithms: Simulations on the complete sequence of small proteins. *Proteins: Structure, Function, and Genetics*, 29:179–184, 1997.
- [50] Jan T. Pedersen and John Moult. Protein folding simulations with genetic algorithms and a detailed molecular description. *J. Molecular Biology*, 269:240–259, 1997.
- [51] L. Pela, J. Kostrowicki, and H. A. Scheraga. The multiple-minima problem in the conformational analysis of molecules. deformation of the potential energy hypersurface by the diffusion equation method. *J. Phys. Chem.*, 93:3339–3346, 1989.
- [52] RCSB protein data bank. <http://www.rcsb.org/pdb/>.
- [53] I. Rechenberg. *Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.
- [54] Jr. Richard W. Burkhardt. *The spirit of system : Lamarck and evolutionary biology*. Cambridge, Mass. : Harvard University Press, 1977.
- [55] Frederic M. Richards. The protein folding problem. *Scientific American*, January 1991.
- [56] M J Rooman, J-PA Kocher, and S J Wodak. Prediction of protein backbone conformation based on seven structural assignments. *J. Mol Bio*, 221:961–979, 1991.
- [57] B. Rost and C. Sander. Prediction of protein secondary structure at better than 70% accuracy. *Journal of Molecular Biology*, 232(2):584–599, 1993.
- [58] Ayed A. Salman, Chilukuri K. Mohan, and Kishan Mehrotra. Adaptive linkage crossover. *Evolutionary Computation*, 8(3):341–370, 2000.
- [59] Steffen Schulze-Kremer. Genetic algorithms and protein folding.

- [60] Steffen Schulze-Kremer. Genetic algorithms for protein tertiary structure prediction. In R. Manner, , and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, 1992.
- [61] Steffen Schulze-Kremer. Application of evolutionary computation to protein folding with specialized operators. In *Evolutionary computation in bioinformatics*, pages 163–191. Morgan Kaufmann, Amsterdam ; New York, 2003.
- [62] Claude Elwood Shannon. *The mathematical theory of communication*. University of Illinois Press, Urbana, 1949.
- [63] Alena Shmygelska, Rosala Aguirre-Hernandez, and Holger H. Hoos. An ant colony optimization algorithm for the 2d hp protein folding problem. In *Lecture Notes in Computer Science*, volume 2463, page 40 ff., 2002.
- [64] R.C. Stevens. The cost and value of three-dimensional protein structure. *Drug Discovery World*, 4:35–48, 2003.
- [65] S. Sun. Reduced representation of protein structure prediction: statistical potential and genetic algorithms. *Protein Sci.*, 2:762–785, 1993.
- [66] IBM Blue Gene Team. Blue gene: A vision for protein science using a petaflop supercomputer. *IBM Systems Journal*, 40(2):310–318, 2001. <http://www.research.ibm.com/bluegene/>.
- [67] R. Unger and J. Moult. A genetic algorithm for three dimensional protein folding simulations. In *5 th Intl. Conf. on Genetic Algorithms*, pages 581–588, 1993.
- [68] R. Unger and J. Moult. Genetic algorithms for protein folding simulations. *J. of Molecular Biology*, 1(231):75–81, 1993.
- [69] Sandor Vajda and Charles Delisi. Determining minimum energy conformations of polypeptides by dynamic programming. *Biopolymers*, 29:1755–1772, 1990.
- [70] M. Vázquez and H. A. Scheraga. Use of buildup and energy-minimization procedures to compute low-energy structures of the backbone of enkephalin. *Biopolymers*, 24:1437–1447, 1985.

- [71] Kanta Vekaria and Chris Clack. Biases introduced by adaptive recombination operators. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 1, pages 670–677, Orlando, Florida, USA, 13–17 1999. Morgan Kaufmann.
- [72] Cameron Wellock and Brian J. Ross. An examination of lamarckian genetic algorithms. In Erik D. Goodman, editor, *2001 Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 474–481, San Francisco, California, USA, 9–11 2001.
- [73] Anthony White and Franz Oppacher. Adaptive crossover using automata. In *LNCS – PPSN III*, volume 866, 1994.
- [74] Anthony R. P. White. Integrating automata with genetic algorithms in order to provide adaptive operators. Master’s thesis, Carleton University, 1993. M.C.S. Thesis.
- [75] Ying Xu, Dong Xu, and Harold N. Gabow. Protein domain decomposition using a graph-theoretic approach. *Bioinformatics*, 16(12):1091–1104, 2000.
- [76] Mutsunori Yagiura and Toshihide Ibaraki. Use of dynamic programming in genetic algorithms for permutation problems. *Eur. J. Oper. Res.*, 92(2):387–401, 1996.
- [77] Bojan Zagrovic, Christopher D. Snow, Michael R. Shirts, and Vijay S. Pande. Simulation of folding of a small alpha-helical protein in atomistic detail using worldwidedistributed computing. *Journal of Molecular Biology*, 323:927–937, 2002. <http://folding.stanford.edu/>.
- [78] Geoffrey Zubay. *Biochemistry*, chapter 1. Addison-Wesley, 1983.