Without inspiration the best powers of the mind remain dormant, there is a fuel in us which needs to be ignited with sparks.

– Johann Gottfried Von Herder (1744 – 1803)

Bio-Inspired Optimization & Sampling Technique for Side-chain Packing in MCCE

Pascal Comte, BSc. Mathematics & Computer Science (Hons.)

Computer Science

Submitted in partial fulfillment
of the requirements for the degree of

Masters of Science

Faculty of Computer Science, Brock University
St. Catharines, Ontario

© April, 2010

Abstract

The prediction of proteins' conformation helps to understand their exhibited functions, allows for modeling and allows for the possible synthesis of the studied protein. Our research is focused on a sub-problem of protein folding known as side-chain packing. Its computational complexity has been proven to be *NP-Hard*. The motivation behind our study is to offer the scientific community a means to obtain faster conformation approximations for small to large proteins over currently available methods. As the size of proteins increases, current techniques become unusable due to the exponential nature of the problem. We investigated the capabilities of a hybrid genetic algorithm / simulated annealing technique to predict the low-energy conformational states of various sized proteins and to generate statistical distributions of the studied proteins' molecular ensemble for pKa predictions. Our algorithm produced errors to experimental results within acceptable margins and offered considerable speed up depending on the protein and on the rotameric states' resolution used.

Acknowledgments

I would like to thank the Dean of Science Dr. Ian Brindle and Dr. A. Joffre Mercier for their support in my inter-disciplinary research endeavours in the fields of Computer Science and Biology. I believe in a strong inter-departmental unity and collaboration. As such, I have been extremely pleased with the ability to conduct research in both fields. This opportunity has promoted further inter-departmental collaborations and has contributed to strengthen the prospects of Brock University.

I would also like to take this moment to express my gratitude to Dr. Doug Bruce of the Department of Biological Sciences for his continuous support and trust in my abilities to conduct research in a field of science unfamiliar to me three years ago. I would like to dedicate special thanks to Dr. Sergei Vassili'ev of the Department of Biological Sciences whose ability to learn and teach concepts of physics, mathematics, chemistry, computer science and biology surpassed my expectations. Working closely with him over the past three years has been a privilege and one unforgettable experience. In the Computer Science department, I would like to give my sincere thanks to Dr. Sheridan Houghten for her guidance and help throughout this research. Finally, I would also like to thank Dr. Brian Ross and Dr. Beatrice Ombuki.

P.C.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The aim of this thesis is to refine part of a search method used in a bio-molecular software package called Multi-Conformer Continuum Electrostatics (MCCE). MCCE was developed in the Gunner Lab at the Physics Department of New York City College. Its use is targeted at the prediction of pH dependent protein properties and can also be utilized as a homology modeling tool. The presented thesis project is in joint collaboration with New York City College. In this Chapter, we first present an introduction to biology terms and theory. We will then define our problem definition and finally, we will elaborate on Computer Science theories specific to our problem.

1.1 Biology Background

A *protein* is a sequence of connected amino-acid *residues*. There are twenty different basic amino-acid residues. In a protein, individual amino-acids are tied to each other forming a long chain. Connectivity is enabled by a chemical peptide bond which occurs when two molecules react with each other. An amino-acid *residue* consists of a backbone and a side-chain. The backbone is made up of a periodically repeating structure of one nitrogen (*N*), two carbons (*C*), two hydrogens (*H*) and one oxygen (*O*). A side-chain is made of a series of connected atoms as shown in Figure 1.1.

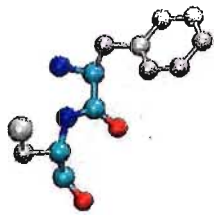

Figure 1.1: Two amino-acid *residues* (Cysteine and Phenylalanine). Backbones are green blue and red atoms. Side-chain atoms are in silver.

Residues only differ in their side-chain composition. The atoms of a side-chain can be in different geometric arrangements. Such a spatial structure can be thought of as a unique side-chain state and is called a *conformer*. The side-chain of a residue can be in only one state at a time. At a maximum, a side-chain can contain five rotatable bonds as shown in Table 1.1.

Table 1.1: Twenty amino-acids and their corresponding number of dihedrals.

Amino-acid	# of bonds	Amino-acid	# of bonds
Alanine (ALA)	0	**Leucine (LEU)**	2
Arginine (ARG)	5	**Lysine (LYS)**	4
Asparagine (ASN)	2	**Methionine (MET)**	3
Aspartic Acid (ASP)	2	**Phenylalanine (PHE)**	2
Cysteine (CYS)	1	**Proline (PRO)**	1
Glutamic Acid (GLU)	3	**Serine (SER)**	1
Glutamine (GLN)	3	**Threonine (THR)**	1
Glycine (GLY)	0	**Tryptophan (TRP)**	2
Histidine (HIS)	2	**Tyrosine (TYR)**	3
Isoleucine (ILE)	2	**Valine (VAL)**	1

There are two types of conformers: *rotamers* and *ionization conformers* or *protonation states*. A conformer is a *rotamer* when a rotation in its side-chain is made about a specific bond point. Rotations of any angular increments can be made. For the sake of computational complexity, rotation increments are normally restricted to be fairly large, ranging anywhere from 30 to 60 degrees. A conformer is an *ionization* conformer when it gains or loses a proton. Figure 1.2 shows the initial conformation of a residue and two of its generated rotamers, achieved using 30 degree increments.

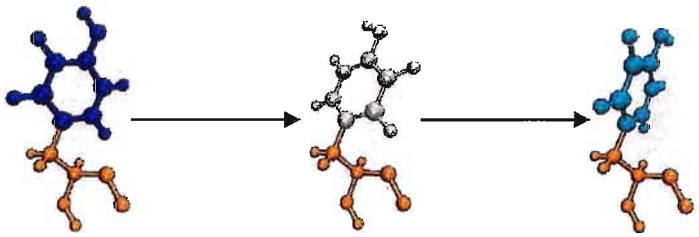

Figure 1.2: Side-chain of Tyrosine (TYR) rotated twice at the top-most rotatable bond. Fixed backbone atoms are in orange.

In general, biologists are interested in understanding the function(s) of proteins. The function(s) of a protein and the chemical properties it possesses are influenced by the protein's electrostatic potential. The electrostatic potential arises from all the charges in the protein. The charged state is found by obtaining the most probable protonation states (*ionization* conformers) of the protein. For example, in photosynthesis, it is worth noting that the electrostatic potential affects the level of light absorption. Moreover, it determines the effectiveness and the rate at which H_20 is oxidized; the amount of oxygen created over a period of time. Now that we have defined essential biological terms subsequently used in this thesis, we, like biologists, are also interested in understanding the function(s) of proteins. Moreover, our interest arises from a complex computational problem which we define in the next section.

1.2 Problem Definition

The goal of this project is to investigate the capabilities of a heuristic search algorithm to predict the most likely low energy conformation of various sized proteins. Our problem is two-fold. First, we need to search through a large search-space for rotamers of individual amino-acid residues that have low energy contributions to obtain a near-optimal protein conformational state. We refer to this computational search problem as *side-chain packing* and it has been shown to be *NP*-Hard [1]. Its search-space is known as a *conformational space* and contains possible states a protein can be in. Secondly, temperature adds kinetic energy to the total energy of a protein, allowing more than one optimal conformation to coexist in the molecular ensemble. Therefore, we are also interested in sampling as many as possible rotamer states for all amino-acid residues of a

given protein, falling within a certain energetic range from the near-optimal solution. More details are given in Section 4.3.1.

A conformational search for a studied protein begins from an initial conformational state, which very often, is taken from a crystallographic x-ray structure or can be experimentally designed. If taken from an x-ray structure, the structure may be obtained at very low temperatures. A low temperature slows down a protein's atoms' movements thus freezing the atoms in place. X-rays are then diffracted through the protein crystal. It then takes modeling to transfer the diffracted pattern to get the atomic positions. Obtaining the low energy states of a protein is essential for calculations of their electrostatic potentials. This data cannot be obtained from the crystallographic x-ray structure, which generally only shows one position for side-chains' atoms. Low energy states can be found with the help of computer search techniques. We emphasize that *side-chain packing* is not a *protein folding* problem. *Protein folding* is the problem of establishing optimal positions and orientations of a protein's backbones and side-chains. This problem is very hard and high-dimensional, making it difficult to approximate a solution [2]. Figure 1.3 shows a folded protein backbone and its initial state conformation.

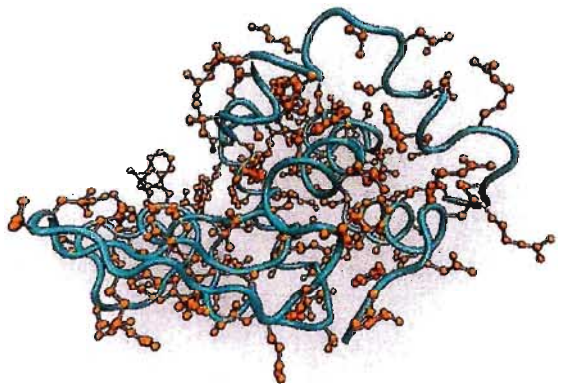

Figure 1.3: Atomic structure of the egg white lysozyme protein (4LZT).
Green helix showing the backbone and initial state side-chains are in orange.

Side-chain packing, on the other hand, keeps the backbone fixed. Our interest lies in the reduced problem of the organization of the side-chains rather than the orientation and

position of their backbones, although *side-chain packing* remains a very hard problem [1]. To better understand some of the issues of searching that emerged from our problem definition, we expand on Computer Science theories. We give an overview of searching, complexity and optimization theory in Section 1.3 and describe several search techniques in Section 1.4.

1.3 Computer Science Background

1.3.1 Searching

In Computer Science, searching is one of the most important problems. Examples of searches include searching strings of text for specific pattern(s), using a website query tool such as Google to search for various information, or simply looking for the shortest path from point A to B under given constraints that minimizes or maximizes a variable of interest. The latter search example falls in the category of optimization problems.

Sometimes, the search is not focused on obtaining an exact solution (optimal) but rather on an approximation of such a solution (near-optimal) due to the large number of solutions to examine. Searches are bound to a *search-space* which represents the set of all solutions for a given problem. The number of solutions in a search-space is problem specific and can be approximated or exactly calculated, depending on the situation. Search-spaces that are too large can cause exhaustive searches to be too time consuming to complete in reasonable time. A search that does not make use of problem specific information can be thought of as an *uninformed* or a *naive* search. Optimization techniques that use problem specific information can be thought of as *informed searches* and are used to *intelligently* guide the process of searching through large search-spaces, avoiding a time consuming enumeration of all solutions.

Performing an informed search *may* possibly limit the algorithm from finding the optimal solution leaving only near-optimal solutions to be found. However, this type of search is not restricted from exploring areas of the search-space far away from the optimal solution or near-optimal solutions. An informed search, also known as a *heuristic* search

algorithm, helps reduce search time while possibly lowering the quality of the solution. In general, a heuristic algorithm approaches optimality but it *cannot guarantee optimality*. In some special cases of heuristic algorithms, optimality of solution can be guaranteed.

For example, consider the A* best-first graph heuristic algorithm [3]. This algorithm finds the least-cost path from a starting node to a goal node from a set of nodes in a graph. The heuristic cost function $f(x)$ is defined as the sum of two functions: $g(x)$ is the path-cost from the initial node to the current node and $h(x)$ is the *heuristic estimate* of the distance to the goal node. The latter function must not overestimate the distance to the goal. The algorithm can guarantee optimality only if the condition $h(x) \leq d(x,y) + h(y)$ is met for every edge (x,y) in the graph. Suppose that $d(x,y)$ is the length of edge (x,y). Then, the A* algorithm is equivalent to Dijkstra's algorithm [4] which guarantees the optimal solution for the shortest path problem of a graph.

1.3.2 Complexity Theory

In Computational Complexity Theory, problems are grouped into classes according to their difficulty. A problem is deemed to belong to class *P* (polynomial) if there is at least one known algorithmic solution which can execute in polynomial time. That is, the execution time of the algorithm is bounded by a polynomial of the size of the problem input. A problem belongs to class *NP* (non-deterministic polynomial) if its solution can be verified in polynomial time. The class of problems *P* is a subset of the class of problems *NP*. In the worst case scenario, a solution to an *NP* problem will require an exhaustive search. An *NP*-Hard problem is one that is at least as difficult to solve as the hardest problems in class *NP* but that does not necessarily belong to class *NP*.

1.3.3 No Free Lunch Theorems

In their research entitled the "No Free Lunch Theorem" (NFL) for search/optimization problems in 1997 [5], D.H. Wolperts and W.G. Macready showed that algorithms searching for a global minimum or maximum of a cost function perform statistically exactly the same when averaged over all possible cost functions, assuming no prior knowledge. It is theoretically impossible to construct a universal search strategy for all optimization problems. Their theorems established that for any algorithm, any increased performance over one class of optimization problems is precisely paid for in performance over another class. Furthermore, no search algorithm is better than random search, over all possible cost functions. These theorems also resulted in a direct analysis of what it means for an algorithm to be well suited to a specific optimization problem [5]. A paper authored by J. Culberson [6], further expanded on the work of D.H. Wolperts and W.G. Macready's original theorems by suggesting that evolution of compound systems exhibiting high degrees of orderliness is not equivalent in difficulty to optimizing hard (complex) problems. Also, the confidence in genetic algorithms as universal optimizers is not justified by natural evolution.

1.4 Search Techniques

In artificial intelligence, there are many types of search methods which behave non-deterministically. That is, given an input, an algorithm can produce a different output result each time. More precisely, at any point, the current state of such a process cannot completely determine the next state. These types of algorithms are also known as *stochastic* processes. Since side-chain packing belongs to the class of *NP*-Hard problems, we give an overview of a few such stochastic search methods which could potentially be well suited for the problem of side-chain packing as described in Section 1.2.

1.4.1 Genetic Algorithm

Genetic algorithms (GA) use *natural* techniques borrowed from biological evolution to find an acceptable solution to computational problems. The concept relies on using an abstract *chromosome* data structure that contains a fixed number of genes. An optimization problem is usually encoded using an efficient mapping between the genes of the chromosome to an actual solution of the problem. The algorithm then evolves this set of abstract chromosomes, generally using crossover and mutation, over several generations to find a near-optimal solution. The search is driven by a heuristic evaluation function which decides on the fitness of a given chromosome. It is worth noting that in general, a heuristic evaluation function may not encompass all of a problem's details. Consequently, the search may be prevented from performing as advantageously as it could otherwise. More details of this heuristic method are given in Section 2.1 of Chapter 2.

1.4.2 Genetic Programming

Similarly to genetic algorithms, genetic programming (GP) also uses techniques borrowed from biological evolution. Unlike GA however, a basic GP defines chromosomes using a tree-based representation. In contrast with the chromosome solution representation of genetic algorithms, GP chromosomes are executable computer programs. These computer programs are also evolved using biologically derived operations such as crossover and mutation. Due to the more complex tree-based representation of chromosomes in GP, these operations tend to be more involved. A GP algorithm tends to run much slower than a GA due to the large overhead in parsing the tree-based chromosomes to execute the programs. Tree-based genetic programming was originally proposed by N. L. Cramer [7] but John R. Koza [8] greatly expanded on this work and patented GP in 1990. More recent studies of Wolfgang Banzhaf et. al. [9] have suggested a linearization of the GP tree-based model, known as linear GP. In this model, a chromosome tree-based representation is flattened out to a similar GA chromosome representation. The linear GP model has also given opportunities for more elaborate and faster algorithm implementations than its former tree-based model [10].

1.4.3 Simulated Annealing

Another stochastic search technique known as simulated annealing (SA) can be used for solving low and high-dimensional problems. Similar to genetic algorithms, the success of this method relies on the *natural* physical process of annealing. Its use on digital computers was proposed by S. Kirkpatrick et. al. in 1983 [11] and by V. Cerny in 1985 [12]. The process of annealing is used in the forging of metals and glass. The reliability of simulated annealing is dependent upon user adjustable input parameters. A problem specific encoding to a generic solution also needs to be determined. Unlike a genetic algorithm, basic simulated annealing optimizes a single solution per generation instead of a pool of potential candidate solutions. New solutions are found by introducing perturbations in the neighbourhood of the current solution. The acceptance of new solutions is based on an exponential probability analogically to the physical process of annealing. More details of this heuristic method are given in Section 2.2 of Chapter 2.

1.4.4 Tabu Search

Tabu search is a *metaheuristic* algorithm that falls in the category of local search techniques and was first introduced in 1989 by F. Glover [13] [14]. The goal of a *metaheuristic* algorithm is to solve a general set of computational problems by combining several heuristic functions. A Tabu algorithm searches a search-space by iteratively moving from a solution A to a solution B, in the neighbourhood of the solution A. The method relies on heavy use of long-term memory structures (a tabu list) to keep track of visited solutions by constantly modifying the neighbourhood list for each solution as the search is performed. In addition, the technique also keeps track of recently visited solutions in a short-term memory structure to avoid visiting redundant search-space areas.

1.4.5 Particle Swarm Optimization

Particle swarm optimization (PSO) is a fairly recent stochastic search method first introduced in 1995 by J. Kennedy and R. C. Eberhart [15] [16]. This technique is borrowed from *naturally* emerging social behaviour noticeably observable in for instance, swarms of birds and bees. This type of social behaviour is referred to as swarm intelligence. The goal of this search methodology is to find a solution to an optimization problem and it can also be used to model social behaviour for different environment settings with multiple constraints. A swarm is modelled by abstract particles in a high-dimensional space that have both position and velocity parameters. Particles are cognitively aware of their performance by keeping track of both their own best position and the global best position in the search-space. Particles' velocities are large when the search first begins and slow down as particles near the global optimal solution.

1.4.6 Monte-Carlo

The Monte-Carlo method is a search technique whereby continuous *random* sampling of its results is performed in order to obtain an approximation to a computational problem that has no known deterministic algorithm to solve it. It was first proposed by S. Ulam in the mid 1940's [17]. Monte-Carlo sampling is extremely useful for problems with high coupled degrees of freedom, for example in weather forecasting and bio-molecular simulations. Applications of Monte-Carlo rely heavily on random numbers. *Truly* random numbers are not always required to be effective. Though the quality of a pseudo-random number generator usually gives an idea of how well a Monte-Carlo simulation will perform.

1.5 Thesis Organization

Several studies, as we will discuss in Section 3.2 of Chapter 3, have revealed that genetic algorithms and simulated annealing are independently very well suited for side-chain packing. To help with our first problem of side-chain packing, we investigated the extent to which a combination of both GA and SA (a hybrid) would perform. Scientists in many fields have recognized that hybrid heuristics can search for some problems' near-optimal solutions more effectively than individual heuristic algorithms [18] [19] [20] [21], but new applications have yet to take advantage of this research. Using our hybrid GA/SA approach, we also developed a novel evolutionary sampler. It helped us with the second problem of obtaining several unique rotamer states for residues falling within a certain energetic range from the near-optimal solution. Additionally, MCCE relies on a Monte-Carlo sampling for the pKa prediction of amino-acid residues. Although this algorithm falls outside the scope of this thesis, it remains an important and required step of MCCE. More information is given in Section 3.1 of Chapter 3.

The remainder of this thesis is organized as follows. Chapter 2 gives a clear and detailed description of genetic algorithms and simulated annealing with examples. Chapter 3 looks at previous heuristic search techniques applied to side-chain packing. Chapter 4 introduces our hybrid GA/SA heuristic and evolutionary sampler methodologies and describes them in detail. Chapter 5 discusses our analysis on 24 different proteins ranging from 52 to 182 residues. We also illustrate the amount of computational time required compared to the current method of MCCE and show various statistical results of performed benchmarking tests against published results of MCCE [22] and experimental data.

Chapter 2

Details of a Genetic Algorithm and Simulated Annealing

The goal of this chapter is to facilitate the reader's assimilation of two fundamental concepts that are used in this project. We cover the inner workings of a generic genetic algorithm and a simulated annealing algorithm. Both naturally occurring physical processes can be implemented as computer algorithms and used for the search of solutions to various optimization problems. In both cases, we demonstrate the examined tactics using the Travelling Salesman Problem (TSP).

2.1 Biology & Genetic Algorithms

In Biology, Genetics is the study of how information contained in chromosomes is transmitted from one generation to the next. The genetic material of living organisms is encoded as a linear sequence of deoxyribonucleic acid (DNA) called chromosomes. The mechanism of genetic inheritance involves reproduction. Genetic diversity of chromosomes arises from the shuffled exchange of DNA between homologous chromosomes during the process of meiosis. This exchange is known as crossing over. Changes can also occur in chromosomes due to mutations, which are rare spontaneous alterations in the DNA sequence. Although mutations do not solely drive the evolution of chromosomes, they do provide the necessary genetic variety upon which natural selection acts. Some chromosomes may confer better suitability to an organism to external selection pressures. Those that do are usually more likely than others to pass

their genes to the next generation. The heart of a genetic algorithm lies in the idea that genetic information in a given generation is exchanged among fitness proportionally selected parent chromosomes to produce the next generation.

2.1.1 Genetic Algorithms on Digital Computers

A computer *genetic algorithm* is a search method in which the basic mechanisms are derived from biological evolution. Simulating evolution on digital computers was first suggested by the geneticist Alex Fraser in 1957 [23]. He later published his research findings [24] in 1970. It was not until John Holland's work in the 1970s that a genetic algorithm as an optimization technique developed and offered the milestone of modern genetic algorithms [25]. GAs have ever since been widely adopted in many fields of science and research including in the design of industrial and commercial products. Some may believe that a genetic algorithm is the perfect tool for all search based problems. We remind the reader that it is not a universal search technique justified by means of natural evolution, as duly noted in Section 1.3.3.

On a digital computer, we represent a chromosome by an *array*; a well known linear computer data structure. A chromosome can be thought of as a linear sequence of linked boxes, each box being a different gene. Chromosomes can be represented as binary arrays as shown in Figure 2.1, and in this case operate in the search-space $S = \{0,1\}^{C*n}$, where n is the number of genes, C is the number of bits for a gene and $C * n$ represents the chromosome *length*.

Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	Gene 8
000	**110**	**001**	**101**	**111**	**001**	**011**	**100**

Figure 2.1: Binary Chromosome Representation for "000110001101111001011100"
$C = 3, n = 8$; *chromosome length* $= 24$

Although very successful genetic algorithms can be written using a binary representation, many computational problems are encoded using different representations. A genetic algorithm may run faster and be more convenient to program for a specific problem if its chromosome encoding is such that it operates in a search-space $S' \subset \mathbb{Z}^n$ or $S' \subset \mathbb{R}^n$

instead of $\{0,1\}^{Cn}$. In fact, real-coded (chromosome encoding operating in $S' \subset \mathbb{R}^n$) genetic algorithms search for better solutions and are more suited for nonlinear high-dimensional optimization problems than binary genetic algorithms [26]. An example of an integer-coded chromosome encoding of Figure 2.1 is given in Figure 2.2.

Gene 1	Gene 2	Gene 3	Gene 4	Gene 5	Gene 6	Gene 7	Gene 8
0	6	1	5	7	1	3	4

Figure 2.2: Integer-coded Chromosome "06157134" of the Binary-coded Chromosome "000110001101111001011100"

While a genetic algorithm usually holds a single pool of chromosomes, other implementations exist that use more than one pool. Once a chromosome mapping is defined, the next step is to construct a set of genetic operators. Genetic operators allow for manipulation of individuals in different ways and together they form the core of the genetic algorithm. These operators will drive the evolution of the chromosome population over a fixed number of generations. Figure 2.3 depicts the flow of a generic genetic algorithm.

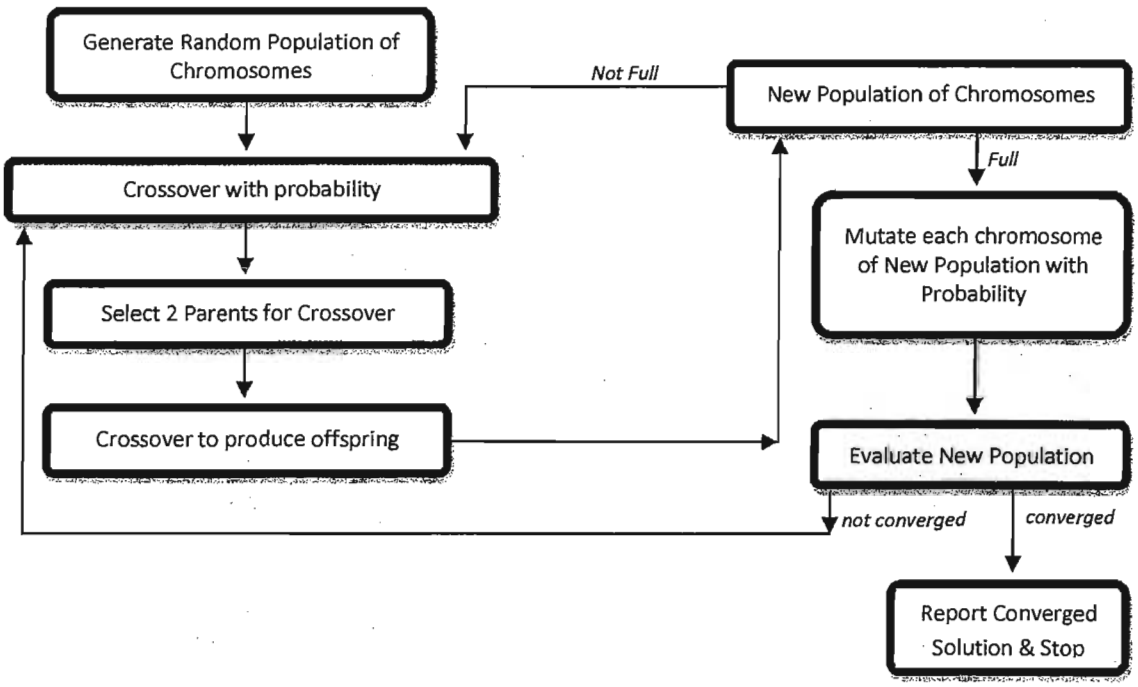

Figure 2.3: Generic genetic algorithm flow

2.1.2 Genetic Operators

Genetic operators are the work horses of the evolutionary algorithm. There is a strong inter-dependence between the different operators. The direct and indirect effects an operator has upon another is still not well understood. However, the role of all the genetic operators are well understood. For simplicity of concept explanation, the operators will be described using a binary chromosome encoding.

Evaluation Function: The evaluation function is not a genetic operator *per se*. It provides a means to establish a fitness value for all chromosomes of a population. This heuristic function is problem specific and sometimes may not grasp all the details of the problem. In general, this is the most computer intensive and time consuming part of the genetic algorithm. The goal is usually to optimize the value of a fitness function to obtain an approximation to a global optimum or an exact optimal solution.

Selection (reproduction) Operator: This operator is mostly associated with the decision of which two parent chromosomes from the population will mate to produce one (or two) offspring. Ultimately, the operator draws a winner, based on the fitness value of the individual. A few well known selection methods include *k-Tournament selection*, *Roulette Wheel selection* and a *naive random selection*.

A k-Tournament selection method selects k individuals from a population at random in the range $[1..N]$, where N is the total number of individuals in the population. It then retains the one that has the best fitness out of those k chromosomes.

Roulette Wheel is also known as fitness proportionate selection. A weight is attributed to each member of the population based on its fitness value. Given an individual i with fitness F_i, its probability of being selected becomes $P_i = \frac{F_i}{\sum_{j=1}^{N} F_j}$, where N is the population size.

The random selection scheme picks a random individual in the range $[1..N]$. It is a very loose technique and does not usually yield acceptable results since no fitness selective pressure constraint is applied. It is the fastest to implement and can be useful for debugging or testing the validity of other selection methods.

Crossover Operator: This operator creates the essential genetic diversity for evolution of chromosomes by combining shuffled genes of two selected parents. There exist several flavours of this operator, each having its advantages. Usually a crossover operation requires two different parents. Asexual reproduction is possible and it can be a successful operator for some optimization problems. Before a crossover operator generates an offspring, a decision with a given probability is made as to whether these two individuals should reproduce. Crossover rates generally in the 50-95% range. The two most widely used are the *N-point* and *Uniform* crossovers.

In an N-point crossover, N cutting-points are randomly selected. The two parents' genes are swapped between these cutting-points, producing one (or two) offspring. This is illustrated in Figure 2.4 using a 2-Point crossover operator.

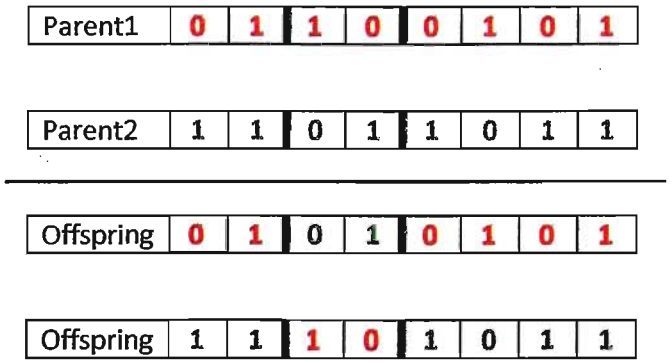

Figure 2.4:

In a Uniform crossover, each gene of the two parents is swapped according to a randomly generated selection mask. The mask gives each gene a 50% probability of being swapped. Figure 2.5 shows this crossover to produce two offspring. In the mask, a zero means a gene swap is made, and a one means no swap occurs. To generate a second offspring, we invert the selection mask.

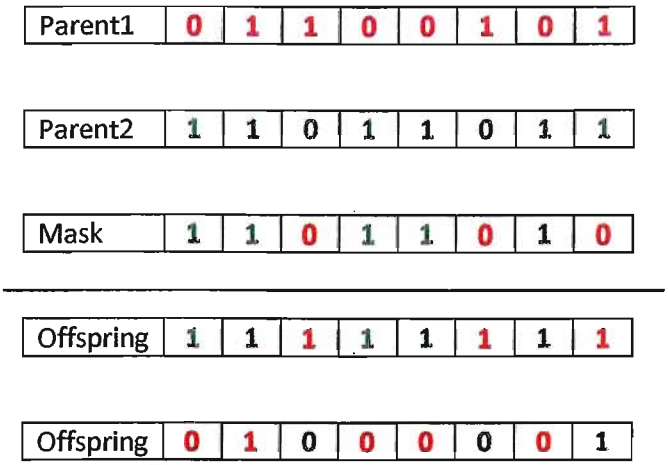

Figure 2.5: Uniform crossover

Mutation Operator: Mutation also allows for genetic diversity by tweaking existing chromosomes. It is worth mentioning that some evolutionary algorithms only make use of a mutation operator and no crossover operator and vice-versa. Both methods have their applicability, advantages and disadvantages. In a generic genetic algorithm, the rule of thumb is to use both crossover and mutation operators, with mutation occurring at a much lower probability than crossover. Two common mutation operators include *single-point* and *multi-point* gene mutations. In a single-point gene mutation, a randomly selected gene has its value changed as shown in Figure 2.6.

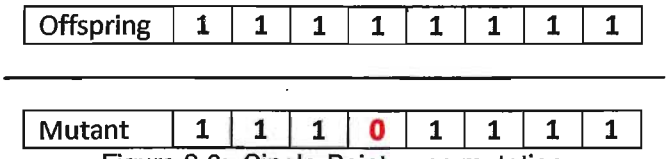

Figure 2.6: Single-Point gene mutation

In a multi-point gene mutation, a random roll with very low probability is taken for each gene to be mutated. If the roll is successful for that gene, it has its value changed as shown in Figure 2.7.

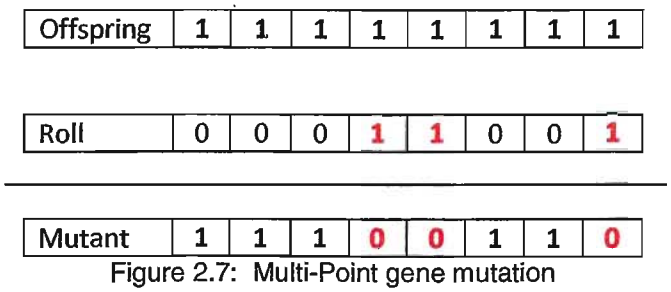

Figure 2.7: Multi-Point gene mutation

Elitism Operator: Although this is a specialized operator, it is often referenced in literature as the one that allows the search to converge to near-optimal solutions. Some argue that it may get the algorithm stuck at a local optimum (pre-mature convergence) from which it will not be able to escape. If used with care, one may find it very useful. In general, a very small number of solutions (e.g. 1%) with the best fitness are directly copied over in the next population at each generation. The elitism percentage is flexible to the point where only a single best individual might be copied.

Niching Operator: The niching operator is another specialized mechanism to take care of duplicate chromosomes in a population. Large numbers of duplicate chromosomes usually start to appear when the genetic algorithm pre-maturely converges to a local optimum. Instead of removing duplicate chromosomes from the population, the niching operator is used as follows. If a chromosome A is a duplicate of chromosomes B, C and D, then chromosomes B, C and D have their genes mutated at a higher probability than the normal mutation. This operator also encourages further genetic diversity.

2.1.3 Genetic Algorithm Example

In this example, we use an instance of the Travelling Salesman Problem (TSP). The TSP belongs to the class of *NP*-Hard problems [27] and there is no known efficient algorithm for finding the optimal tour. In the context of the TSP, a tour is defined to be an ordered sequence of cities that have to be visited exactly once. A cost is attributed to travel between each city pair. A tour begins at any starting city and ends back at the starting city. Figure 2.8 shows this concept. The goal of the TSP is to find such a tour that minimizes the total cost of travel.

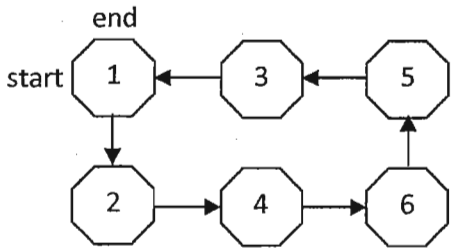

Figure 2.8: A TSP tour, $n = 6$ cities
Search-space = $\frac{6!}{2}$ = 360 possible tours

The tour of Figure 2.8 begins at city 1, visits intermediate cities 2-5 exactly once and comes back to city 1. The total number of choices can be easily calculated as a factorial of the problem size. For n cities, there are $n!$ possible tours. Furthermore, a tour {A, ... , B} has an identical cost of travel to its backward tour {B, ... , A}. Thus, there are $\frac{n!}{2}$ unique possible tours for an n city TSP problem. For a 100 city TSP, there are approximately 4.67×10^{157} possible tours.

Let us assume we wish to visit 10 cities, thus have a TSP problem with $n = 10$. We wish to find the tour that will minimize our total cost of travel given a set of costs. We design a basic genetic algorithm with the aim of approximating an optimal solution to this problem. Table 2.1 shows an example matrix M giving the cost of travel between all city pairs. The lower triangular matrix in bold is simply a reflection about the diagonal axis of zeros. This diagonal indicates that moving from a city to itself is not a legitimate move.

Table 2.1: Matrix of travelling costs between all city pairs

CITY	1	2	3	4	5	6	7	8	9	10
1	0	30	40	24	56	32	12	4	45	9
2	30	0	19	43	21	54	34	32	11	18
3	40	19	0	65	44	32	32	54	23	33
4	24	43	65	0	23	13	51	43	31	29
5	56	21	44	23	0	11	21	31	41	51
6	32	54	32	13	11	0	61	71	31	33
7	12	34	32	51	21	61	0	45	23	35
8	4	32	54	43	31	71	45	0	12	21
9	45	11	23	31	41	31	23	12	0	34
10	9	18	33	29	51	33	35	21	34	0

Chromosome Encoding: The first step is to define a chromosome mapping to represent a tour. Since the TSP is *NP*-Hard, we use an integer-coded chromosome representation instead of a binary encoding. Note that a binary encoding would also be valid. Each

gene represents a city number (integer) to be visited. Since this is an ordered gene problem, no city can occur more than once in a chromosome.

Solution Interpretation: A chromosome tells us the order in which to travel, starting from the city indicated by the first gene of the genetic sequence. The second gene tells us which city to visit next. The last gene of the chromosome is the last city to visit, after which we need to travel back to the starting city designated by the first gene. We show an example candidate solution for our TSP problem in Figure 2.9.

Gene1	Gene2	Gene3	Gene4	Gene5	Gene6	Gene7	Gene8	Gene9	Gene10
1	**6**	**2**	**5**	**7**	**4**	**3**	**8**	**10**	**9**

Figure 2.9: Example of a candidate solution
Tour = {1, 6, 2, 5, 7, 4, 3, 8, 10, 9, 1}

Evaluation Function: We are given the cost of travel for all city pairs in matrix M of Table 2.1. Our function is a sum of accumulated costs from the first city to the second city, from the second city to the third city etc, up to the last city back to the first city. The fitness of an individual i can be expressed as $F_i = (\sum_{i=1}^{n-1} M[Gene[i]][Gene[i+1]]) + M[1][Gene[n-1]]$, where the integer value of $Gene[i]$ is the city number of the i^{th} gene in the chromosome, $M[i][j]$ is the cost found in the matrix for travelling between the city pair $[i, j]$ and $n = 10$. The second term of the equation adds the cost of travelling from the last city back to the first (starting) city. Figure 2.10 shows the concept of calculating the travel cost for the example chromosome given in Figure 2.9. The costs are obtained from the example matrix M in Table 2.1.

Gene1	Gene2	Gene3	Gene4	Gene5	Gene6	Gene7	Gene8	Gene9	Gene10
1	**6**	**2**	**5**	**7**	**4**	**3**	**8**	**10**	**9**
	(6 to 2): 54		(5 to 7): 21		(4 to 3): 65		(8 to 10): 21		
(1 to 6): 32		(2 to 5): 21		(7 to 4): 51		(3 to 8): 54		(10 to 9): 34	

Figure 2.10: Total Cost = 32 + 54 + 21 + 21 + 51 + 65 + 54 + 21 + 34 + (9 to 1: 45) = 398

Crossover operator: A new population of chromosomes is generated by continuously inserting two offspring, resulting from the union of two selected parent chromosomes, until the population reaches its maximum capacity. Parents are not included in the new

population. Asexual reproductions were also forbidden. We used a simple 1-point crossover operator, with 0.9 probability. To take care of the ordered gene constraint, we discarded all offspring that contained duplicate cities in their chromosome sequence.

Mutation operator: A city is allowed to be visited exactly once. This implies that we cannot blindly change a chromosome's gene to a random one. As such, we decided to use a mutation operator with 0.1 probability whereby we swapped two randomly selected genes, and thus kept the ordered sequence constraint of the problem.

Selection operator: We opted for a simple 2-Tournament selection.

Elitism: We decided to keep a single best chromosome at each generation.

Initial Population: We randomly generated an initial population of 50 individuals with the constraint that each city could only appear once in a chromosome.

Optimal Solution: Since the search-space was fairly small (1,814,400 solutions), we were able to generate an enumeration of all the tours. The optimal tour for this problem has a fitness score of 141. Actually, there are 3 different optimal tours, all having the same minimized cost of 141: {3, 2, 9, 8, 1, 7, 5, 6, 5, 10}, {3, 2, 9, 8, 1, 10, 4, 6, 5, 7} and {4, 6, 5, 7, 3, 2, 9, 8, 1, 10}.

Results: Table 2.2 shows the results after executing the genetic algorithm 200 times using different seed values.

Table 2.2: Results of 200 instances of the Genetic Algorithm

Average Fitness of Near-Optimal Solution	162.225
The Best Fitness of Near-Optimal Solution	141
The Worst Fitness of Near-Optimal Solution	204

The genetic algorithm did not produce the same near-optimal solution every time it was executed. In fact, it found an optimal solution a total of 19 times out of 200 instances.

The two solutions found with the optimal cost of 141 were: {4 6 5 7 3 2 9 8 1 10} and {3 2 9 8 1 7 5 6 5 10}.

2.2 Simulated Annealing

The term "simulated annealing" came from the natural process of annealing common in metallurgy and glass making. The idea relies on first heating up a piece of material, up to about 50^{o} above the temperature at which an iron-based metal changes its crystal structure from ferrite to austenite [28]. This temperature is held for a sufficient amount of time for the material to form a grain structure. Then, slow continuous cooling is applied over a fixed period of time to allow the material to reach a crystal structure in an equilibrium state. Applying this process to a material causes some changes in the material's properties such as strength, hardness and structural stress. Cooling is usually applied by means of a cold water tank or cold air. Figure 2.11 shows the annealing of metal sheets which changes the sheets' hardness property, allowing them to be bent more easily.

Figure 2.11: Annealing of metal sheets

A simulated annealing algorithm was first proposed by S. Kirkpatrick et. al. in 1983 [29], and by V. Cerny in 1985 [12]. Analogous to annealing, the idea is to represent the material's physical state as a solution to an optimization problem. The goal of SA is to search for a solution (state) by undergoing refinement at each temperature step. Initially, the simulated temperature is set to a high value and is gradually lowered according to some user defined rule. The initial solution (state) of the optimization problem is generally randomly generated. The SA then generates another state in the neighbourhood of the current solution. The algorithm decides whether to move to this new state using a

probability function based on the current simulated temperature. Better states are always accepted. When the temperature is high, new states that are worse than the current one have high probabilities of being accepted. As the temperature is reduced, the algorithm rejects worse states more frequently, thus accepting only those states that provide an improvement.

The ability to pick a worse state is Simulated Annealing's main strength. It enables searching completely new avenues until a global optimum is approximated or achieved. The global optimum is the target optimal solution of search algorithms. Local optima are wells, defined by the landscape of the problem's search-space. An optimization problem may have several such local optima, as shown in Figure 2.12. In essence, the goal of any heuristic search algorithm is to explore those wells without permanently falling inside one of them.

Figure 2.12: Global and local minima

In 1994, V. Granville et. al. [30] showed that, given a finite problem, the probability a simulated annealing algorithm will find the optimal solution approaches 1 as the execution time of the algorithm is extended. Depending on the problem's search-space cardinality this may be impractical and may require many years of modern computing time. We recall that from the No Free Lunch Theorems, as noted in Section 1.3.3, it is impossible to construct a universal search algorithm for all optimization problems. Therefore, the probability of finding the optimal solution using a Simulated Annealing may *approach* 1 but may very well never exactly reach it.

2.2.1 Simulated Annealing on Digital Computers

Implementing a generic simulated annealing algorithm on a digital computer is simpler than our previously visited genetic algorithm. There are two technicalities which will require our attention. First, we show the flow of a generic SA in Figure 2.13.

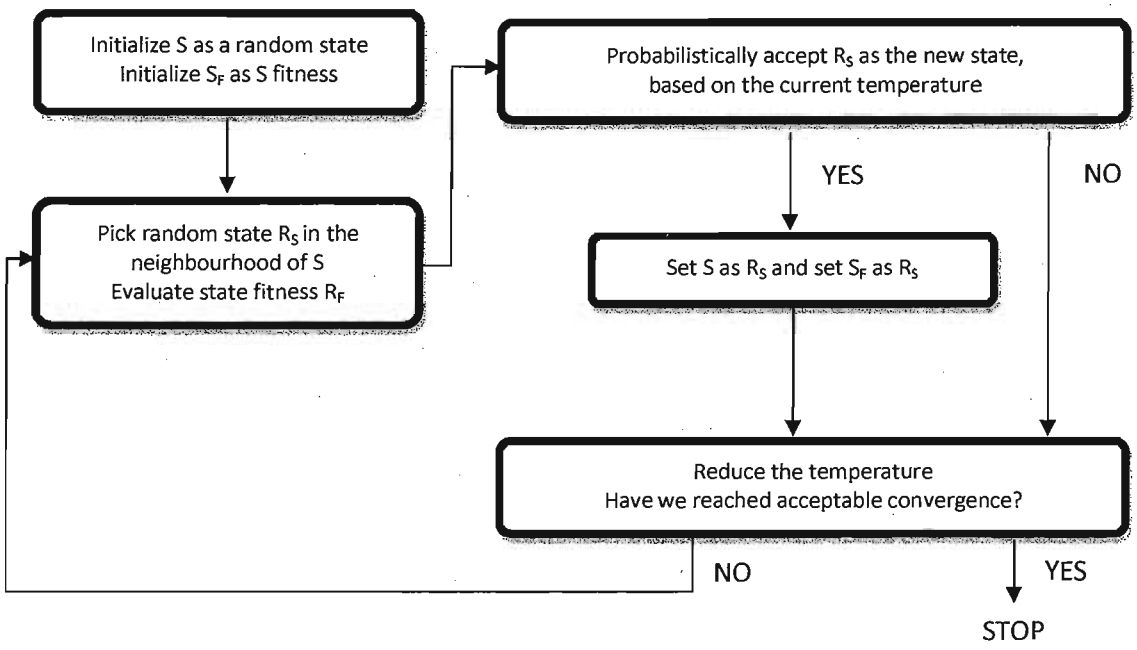

Figure 2.13: Generic simulated annealing flow

Our first point of interest lies in the probabilistic acceptance of a new state. The probability of moving from the current state to a new one is usually based on a decreasing exponential function that relies on the simulated temperature of the system. The probability function may be defined by Equation 2.1.

$$P\left(R_f, S_f, T\left(\frac{\text{current step}}{\text{max step}}\right)\right) = \begin{cases} 1 & \text{if } R_f < S_f \\ e^{(S_f - R_f)/T} & \text{otherwise} \end{cases} \tag{2.1}$$

$T\left(\frac{\text{current step}}{\text{max step}}\right)$ is a user defined temperature function that controls the temperature cooling rate, S_f is the fitness of the current state and R_f is the fitness of the new random neighbour state. If T is large, many bad solutions will be accepted and many new avenues will be explored. If T is small, fewer bad moves are accepted and the search has somewhat converged to what we hope is a global optimum.

The second point of interest lies in the cooling function of the simulated temperature, also known as the annealing schedule. The rate at which the temperature decreases has a severe impact on the performance of the algorithm. Studies have shown that in general, a simple cooling scheme such as $T_{i+1} = \alpha * T_i$ is good enough for most applications, where T_i is the current temperature, T_{i+1} is the next temperature and α is the reduction factor in the range (0.0,1.0).

The initial temperature of the system is actually problem specific and it also determines the effectiveness of the algorithm. Kirkpatrick [29] suggested to use one that results in an average increased acceptance probability of roughly 0.80. We can calculate this initial temperature T_0 by performing a quick search whereby all increases are accepted, where the average objective increase is computed as Δf^+, to obtain $T_0 = -\frac{\Delta f^+}{ln(0.8)}$.

2.2.2 Simulated Annealing Example

We re-visit the previous example of Section 2.1.4 of the Travelling Salesman Problem. We wish to visit 10 cities, each exactly once and come back to the starting city. The cost of travel between each city pair is given by the matrix M of Table 2.1. Our goal is the same as before: to find a near-optimal tour or find an exact optimal tour if possible.

We design a basic simulated annealing algorithm to approximate an optimal solution to this problem. The following are required to implement a simulated annealing algorithm on a digital computer:

1) Initial and Ending temperatures
2) Annealing Schedule function
3) Probability function
4) Problem Representation (encoding)
5) A means to define a random neighbour from a state

Problem Representation: As used in Section 2.1, we represent a TSP with an integer based array data structure. We remind the reader that we are no longer dealing with chromosomes.

Random Neighbour Definition: We define this by swapping any two adjacent cities, keeping the ordered sequence problem constraint.

Probability Function: We will be using Equation 2.1.

Annealing Schedule: We use $T_{i+1} = 0.999 * T_i$, where i is the temperature step.

Initial and Ending temperatures: We perform an initial quick random search to determine a meaningful starting temperature as the literature suggested, for 50 temperature steps of the algorithm. The Simulated Annealing ends when the internal simulated temperature reaches 0.0005.

Optimal Solution: We recall that the cardinality of our TSP search-space is $\frac{10!}{2} = 1{,}814{,}400$. As stated in Section 2.1.3, there are 3 different optimal tours with a cost of 141: {3, 2, 9, 8, 1, 7, 5, 6, 5, 10}, {3, 2, 9, 8, 1, 10, 4, 6, 5, 7} and {4, 6, 5, 7, 3, 2, 9, 8, 1, 10}.

Results: Table 2.3 shows the results after executing the simulated annealing algorithm 200 times using different seed values.

Table 2.3: Results of 200 instances of the Simulated Annealing algorithm

Average Initial Simulated Temperature	100.537
Average Fitness of Near-Optimal Solution	164.545
The Best Fitness of Near-Optimal Solution	141
The Worst Fitness of Near-Optimal Solution	183

These results are encouraging and emphasize the nature of stochastic algorithms. The Simulated Annealing approach found an optimal tour 5 times out of 200 instances and was able to find all 3 different optimal tours. This phenomenon reinforces the fact that it is possible for a problem to have several global optimum solutions. Which one to pick becomes a matter of preference.

Chapter 3

Previous Research

Before thoroughly explaining our methodology in Chapter 4, we feel it is necessary to discuss a few search techniques that have been previously successfully applied to side-chain packing. We first cover MCCE's method in Section 3.1. Section 3.2 will examine the genetic algorithm, simulated annealing, a deterministic algorithm and particle swarm optimization.

3.1 Multi Conformer Continuum Electrostatics

MCCE [22] is a software product developed at the Physics Department of New York City College in Dr. Gunner's Research Laboratory. The program allows for prediction of a protein's conformation at different pH values and gives the user the final pKa values of the protein's residues of interest. This data can be compared against experimental pKa values and usually requires extensive analysis. The software is divided into four distinct phases of execution. Each phase is dependent upon the previous one. The first phase initializes the protein and its parameters. For each amino-acid residue in the protein, a pre-defined list of torsion bond points for its side-chain and a list of possible ionization states (see Section 1.1 of Chapter 1) are loaded. The second phase generates rotamers, performs a random side-chain sampling and inserts ionization conformers based on the rotamers that were sampled. The third phase relies on an external program called *Delphi.* It calculates the electrostatic interactions of the ionization conformers inserted in the previous phase. Ionization conformers are essential for the calculation of electrostatic forces in this third phase and they are specific to each amino-acid residue. Finally, the

fourth phase performs a Monte-Carlo sampling on microstates (protein conformations) using the electrostatic calculations from the previous phase. The Monte-Carlo algorithm randomly samples conformers for each amino-acid residue of the protein to obtain the equilibrium distribution of microstates occupied at a given physiological temperature. A microstate is defined to be a sequence of rotamers, one for each of the amino-acid residue.

Rotamer generation and side-chain sampling occurs in MCCE step 2. Our methodology, as explained in Chapter 4, offers an alternate way to perform this side-chain sampling. In MCCE, there are several details involved in the generation of rotamers. Initially, the rotamers are generated as described in Section 4.1.3 of Chapter 4, and are then inserted in the computer represented protein structure shown in Figure 3.1. A single rotation of a bond generates one rotamer. As explained in Section 1.1 of Chapter 1, there can be many possible combinations from performing a rotation at different bond(s) of a side-chain. This is done for all residues of the protein. Subsequently, the rotamers are then pruned based on high self-energy. The removal of rotamers is based on the overlap of one or more atoms with another rotamer or the overlap with the fixed backbone atoms. Pruning eliminates many non-physiological states for each residue.

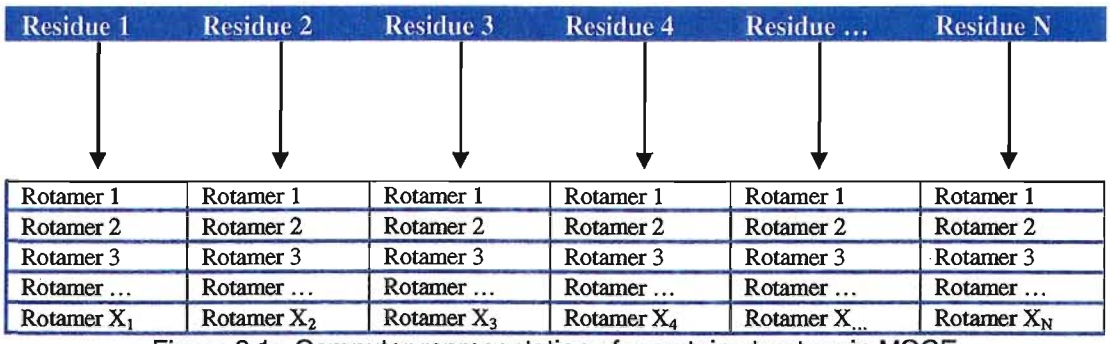

Figure 3.1: Computer representation of a protein structure in MCCE

The remaining rotamers in the protein constitute low self-energy states possible of occurring at physiological temperatures. From this given set of low energy rotamers, MCCE's side-chain sampling algorithm is used to find the rotamers that are most likely to occur for each residue. This is determined by their van der Waals (VDW) energy interactions and also by occupancy.

Micro-states (conformations) that have low energy are remembered and the occurrence of each rotamer for each residue is updated. The occupancy of a rotamer is based on the number of times it has appeared as part of a low energy conformation of a protein, out of all the low energy conformations found in the random search. Microstates (conformations) are randomly generated until all residues are in a low energy conformation (no more changes). This random process is repeated 5,000 times after which rotamers falling outside the occupancy cut-off are removed from the protein structure. Based on these final rotamers, ionization conformers are inserted in the protein structure. MCCE was shown to produce acceptable pKa prediction results for proteins as described in the papers [31] [32] [33] [34] [35] [36] [37].

3.2 Previous Applications of...

3.2.1 Genetic Algorithms

In [38] R. S. Judson, M. E. Colvin, J. C. Meza, A. Huffer and D. Gutierrez investigated three different search methods on a 2-D polymer for which they calculated the analytical global minimum. They used a genetic algorithm, a simulated annealing algorithm and a random search algorithm, independently of each other. A conjugate gradient (CG) routine was also implemented in all 3 cases, to act as a local search once the convergence criterion was met. This local search further optimized the near-optimal solutions. An important outcome of their research was that as the molecule size increased, both GA and SA performed better than the random search.

In 1993, a different research group of R.S. Judson et.al. [39] claimed that a genetic algorithm was the ideal choice of search method for molecules with more than 8 rotatable bonds. Binary-coded chromosome representation was used, along with a roulette-wheel selection method. Their implementation was actually derived from the Genesis [40] GA code. They used a 2-Point crossover operator. For their mutation operator, each gene of a chromosome had an equal probability of 0.001 to have its bit flipped from 1 to 0 or vice-versa. Not the entire population was regenerated at each generation. They also used

an elitist strategy whereby the most fit individual was always copied directly into the next generation without mutation but could still mate. Their population size was about 10 times the number of dihedral angles undergoing optimization, with a maximum of 100 times that number. They used an angular increment resolution of 5 degrees and used 6 bits to represent a single bond. Their results were compared against those of Sybyl's CSEARCH [41] [42] search method, which utilized an increment resolution of 30. Even using a fine resolution of 5 degrees the GA was able to produce near-optimal solutions much faster than CSEARCH. The tree-pruning of CSEARCH, crucial to CSEARCH's algorithm, suffered significantly as the torsion resolution was decreased. CSEARCH was one of the earliest and most well-developed systematic search algorithms of the 1990s. Finally, they suggested that GA be applied to a set of larger molecules other than the 72 molecules picked from the Cambridge Structural Database [43] for their research, to see how far genetic algorithms could be pushed to find low energy conformations.

In a 1997 study [44], a genetic algorithm was implemented for side-chain packing. Unlike previously discussed research, this research group opted for a real-coded genetic algorithm for its encoding of chromosomes. They used a 2-Point crossover operator and a roulette-based selection method, without the use of elitism. The chance of a chromosome to be selected was proportional to the Boltzmann factor of the energy corresponding to the conformation (chromosome) with a "temperature" of 10,000K, which did not exert a bias toward low energy conformations. The low energy conformations were more likely to be selected than higher ones. For the mutation operator, they argued that mutating all the genes (torsion angles) would lose the information in the chromosome. As such, a given gene was mutated at a probability $m = 0.4$ divided by the number of flexible torsion angles in the molecule. New offspring produced by crossover may in fact be worse than their parents. Therefore, the new population was chosen from both the parents and the offspring, using the roulette-based selection method but with a "temperature" of 1,000K, exerting a bias toward low energy conformations. They also used an energy penalty function to prevent these chromosomes of being selected more than once to maintain as diverse a population as possible. Although their target problem was that of unbranched alkanes (PM-toxin A), their GA

could also be used for other molecules. Their results suggested that their genetic algorithm with 2,000 generations, randomly executed several times, was very much more effective than a Monte Carlo search with 10,000 steps. In addition, the size of the population had a considerable impact on the quality of solutions obtained; the larger the better.

In 1998, the research group of Milan Keser and Samuel I. Stupp [45] utilized a binary-coded chromosome representation for the problem of side-chain packing. They also opted for a roulette-wheel selection in conjunction with a fitness scaling operator. The fitness scaling operator allowed for less fit chromosomes to be selected. A random N-point crossover as well as a niching operator were used. This combination of genetic operators allowed their GA to converge to good solutions very fast. Their mutation operator flipped the bit of a randomly selected gene from 1 to 0 or vice-versa. Their niching operator discouraged the introduction of new chromosomes that were too similar to already existing ones in the population. Instead of discarding those too similar solutions, they mutated them at an increased mutation rate. Elitism was not part of their GA although they felt that their results showed that the use of 1% elitism could have improved their final approximation since the best solution was sometimes lost. They used a full generational GA approach. The population size varied but usually remained within 1 to 3 times the length of a chromosome. To represent a single torsion angle, a chromosome's gene contained anywhere from 5 to 10 bits, resulting in varying angular increment resolutions from $360/2^5$ to $360/2^{10}$ degrees. Their GA searched through a floating point torsion space of rotamers (torsion search) without pre-generating rotamers. Tests were conducted on the molecule Tricosane, which is not a protein. It is the equivalent of a protein containing roughly 7-10 amino-acids like Lysine (LYS) and Arginine (ARG). They concluded that their genetic algorithm worked well for their test case of Tricosane as well as for the self assembling of chiral monomer synthesized in their laboratory.

In a more recent study conducted in 2007, a binary-coded genetic algorithm was used to design protein sequences [46]. Instead of generating rotamers by searching through a

torsion space, the group of researchers used a rotamer library to optimize the tertiary structure of the hydrophobic core of Cytochrome b_{562}, containing only 16 amino-acid residues. Their GA population size was of 220 chromosomes. Their crossover operator randomly determined cut points using the *N*-Point cross-over semantics, and new offspring were accepted based on the Metropolis test: if the offspring's fitness was better than its two parents, it was automatically accepted. Otherwise, it was accepted based on the probability function $P = e^{-(G_2-G_1)/KT}$, where $G_2 - G_1$ is the average free energy difference between the parents (index 1) and the offspring (index 2). They also used 10% elitism. The mutation operator randomly selected a residue (gene) in the chromosome sequence and changed it to a random rotamer. At each generation, they applied mutation to 20% of the population. The GA ran until the convergence criterion was met. Convergence usually occurred after 80 generations. Their results showed that their algorithm was able to find sequences very similar to the core structure with energy slightly lower than the native sequence.

In 2008 Marc N Offman, Alexander L Tournier and Paul A Bates used the idea of alternating evolutionary pressure (AEP) in a genetic algorithm for protein model selection [47]. They investigated whether this method would allow a genetic algorithm to explore further afield and potentially find a better near-optimal solution instead of staying within local minima. The principle of AEP had previously only been used to provide theoretical predictions of algorithm performance [48]. In their approach, Offman et. al. substituted a "scored" generation for a number of intermediate "non-scored rounds". In a non-scored round, the population grew linearly and the structures in the ensemble were allowed to sample energetically unfavourable states. Four different benchmarks were used: a GA without AEP and a GA with 1 to 3 "non-scored" rounds between each scored generation. Creating subtle movements in the protein models using the GA with AEP helped to select better models by nudging them to lower energy states. Improvements beyond the best input model were identified in 25% of the GA without AEP and, 31%, 40% and 30% for the GA with AEP using 1 to 3 intermediate "non-scored rounds" respectively. Finally, they suggested that further research be undertaken to further assist conformational search

to recover from falling into local minima, especially so that best models can be consistently selected from the ensemble of structures.

3.2.2 Simulated Annealing

A research study conducted at Stanford University by Christopher Lee and S. Subbiah [49] in 1990 was successful in implementing a simulated annealing algorithm (US Patent 5241470) for the problem of side-chain packing. In contrast with the previous GA research, here an angular increment resolution of 10 degrees was used. Torsion as well as VDW potentials between side-chain atoms and backbone atoms that remained fixed during the simulation were pre-calculated before the annealing process started to reduce execution time of the search. Explicit use of hydrogen atoms was avoided by using an approximation known as a united atom force field. Doing so is common practice in many molecular dynamics simulations. To avoid getting stuck in local minima, they added several features added to their generic simulated annealing algorithm. A typical run of their SA algorithm would generate roughly 2000 different protein conformations. If a single structure, treated as a canonical ensemble, were desired from the aggregate set, one would use a weighted Boltzmann averaging method given by Equation 3.1.

$$E_{av}(\chi_1,\chi_2) = \frac{\sum E_i(\chi_1,\chi_2) e^{\frac{E_i(\chi_1,\chi_2)}{E_{wt}}}}{\sum e^{-E_i(\chi_1,\chi_2)/E_{wt}}} \tag{3.1}$$

E_{wt} is the weighting energy determining how selective the average should be and, $\chi_1 and\ \chi_2$ are the first two torsion angles. For more details, we suggest reading [49]. Their simulated annealing method was originally meant to predict conformations of 5 to 15 residues. They found it useful to predict side-chain packing of nine test proteins ranging from 45 to 323 amino-acid residues. Their results showed that 80-90% of hydrophobic side-chains were correctly predicted within an overall root mean square deviation (RMSD) of 1.77 Angstrom (Å), in contrast with 30-40% for the other methods at the time. Hydrophobic side-chains are responsible for protein folding and protein-protein interactions. The Reversed Phase Chromatography theory [50] suggests that a hydrophobic effect is driven by the loss of hydrogen bonds in water in the presence of a

hydrophobic molecule. Their research concluded that the van der Waals (VDW) potential is the ultimate determinant for the arrangement of side-chains. Their prediction errors were mainly attributed to surface-exposed residues which were poorly constrained.

3.2.3 Deterministic Algorithms

For just over a decade side-chain packing has been known to belong to the class of *NP*-Hard problems [1]. In a recent study of T. Akutsu. et. al. of 2005 [51], this problem was reduced to a maximum edge-weight clique finding problem. To cope with the size of the side-chain packing problem, they decided to generate rotamers for each residue by rotating each residue's side-chain by an interval of $\left(\frac{2\pi}{K}\right), k = 0,..,K-1$, generating $\left(\frac{2\pi}{K}\right)$ conformations for each amino-acid residue. Only the rotations of side-chain atoms along the χ_1 axis (first torsion angle) was considered in order to cope with the capacity of the clique algorithm. For their experiments, they used $k = 18$, producing 20 rotamers for each amino-acid residue. Even though their algorithm predicted the side-chain packing of the 1TDJ protein containing 494 residues in 146 seconds, it is important to note that in general the generation of graphs required a lot more time (non-published). For the 1TDJ protein for example, its graph with 7526 vertices and 26255646 edges was generated. Such a graph represents the search-space for the algorithm to traverse. Many researchers consider a χ (torsion) angle to be correctly predicted if it falls within 40 degrees of the dihedral angle found in the crystal structure. Their algorithm resulted in χ_1 torsion angles prediction errors of only 1-10%. Their results were also superior to those of a branch-and-bound approach. As future work, they suggested dividing proteins into smaller sub-problems and applying the algorithm on those fragments, allowing for much larger proteins to be handled.

3.2.4 Particle Swarm Intelligence

A recent study conducted by Grecia Lapizco-Encinas et.al. [52], extended a generic Particle Swam Optimization (PSO) framework as described in Section 1.4.5 of Chapter 1, to a cooperative combinatorial search strategy. In a generic particle swam optimization problem, the idea relies on a set of agents (particles), each representing a potential solution, while cooperatively searching the solution space of a given problem. Each particle receives feedbacks from its neighbouring particles, as each moves through the search-space and adapts to its landscape. Particles move through the virtual search-space following static particles called attractors. These attractors represent options for a particular component of the problem. A solution is simply assembled by decoding each particle's position into a choice for that solution's component. Each particle remembers at which position it achieved its best performance and, since each particle is also a member of some neighbourhood of particles, it remembers the particle that performed the best in that neighbourhood. A generic PSO approach can only optimize problems in which the components operate in a search-space $S' \subset \mathbb{R}$. Modifications are required to adapt the algorithm to a different domain. In [53] for example, they adapted a generic PSO for binary-coded solution components.

In Grecia Lapizco-Encinas et.al.'s approach, particles were grouped together in sub-swarms, where each sub-swarm represented a candidate solution. This grouping was achieved by evaluating the overall behaviour and state of each particle. Neighbours of a sub-swarm are a set of sub-swarms, representing the cognitive memory of a sub-swarm of possible candidate solutions. Each particle represented a residue while each of the attractors represented a rotamer of that amino-acid residue. To reduce the complexity of the problem to a finite set of possible rotamers, they used Dunbrack's rotamer library [54]. Side-chain packing remained an *NP*-Hard problem however. They compared their results against optimal solutions found using a deterministic method of C. Kingsford et.al. [55]. Their data set contained 27 different proteins ranging from 46 to 221 amino-acid residues. The CCPSO method was able to produce conformations with energy close to the optimal ones with a relative energy (fitness) error $< 4.36\%$ in the worse case.

Chapter 4

Methodology

In this chapter we describe our methodologies for both the hybrid GA/SA side-chain packing algorithm and the evolutionary sampler. We first explain the generation of rotamers in Section 4.1, as implemented by MCCE and upon which our technique relies. We then give an overview of our hybrid algorithm for side-chain packing in Section 4.2. Section 4.3 covers the core of our search algorithm in all aspects, including genetic operators, evaluation function and the evolutionary sampler. Our techniques rely on an extensive list of parameters, each of which will be explained in Section 4.4.

4.1 Rotamer Generation

One of the earliest decisions made while studying this problem was whether to use a rotamer library, use MCCE's implementation of rotamer generation or develop our own search through the torsion space of each residue while optimizing the packing of their side-chains.

4.1.1 Rotamer Library

A rotamer library offers a small set of rotamers for all amino-acid residues. This experimental data is obtained by averaging the most probable observed rotameric states of each residue over several hundred different proteins. In theory, this method can reduce the search space by a magnitude but side-chain packing remains *NP*-Hard nonetheless [56]. Table 4.1 shows the approximated total number of rotamers for all 20 amino-acid residues as available in the Penultimate Rotamer Library in [57].

Table 4.1: Number of rotamers of all amino-acid residues

Amino-acid	# of rotamers	Amino-acid	# of rotamers
Alanine (ALA)	-	**Leucine (LEU)**	2431
Arginine (ARG)	769	**Lysine (LYS)**	984
Asparagine (ASN)	1396	**Methionine (MET)**	472
Aspartic Acid (ASP)	2041	**Phenylalanine (PHE)**	1570
Cysteine (CYS)	280	**Proline (PRO)**	807
Glutamic Acid (GLU)	1339	**Serine (SER)**	2456
Glutamine (GLN)	761	**Threonine (THR)**	2431
Glycine (GLY)	-	**Tryptophan (TRP)**	584
Histidine (HIS)	562	**Tyrosine (TYR)**	1410
Isoleucine (ILE)	1643	**Valine (VAL)**	2626

No rotamers exist for both Alanine and Glycine, since both have zero rotatable bonds as seen in Table 1.1 of Section 1.1. If we were to conduct a study on a protein whose rotamers did not fall in the average considered by this rotamer library, we could find ourselves incorrectly predicting the protein's conformation. Note that after having defined a set a rotamers for a given protein, a search through the set's search-space is still required to find a near-optimal side-chain packing.

4.1.2 Torsion Search

A torsion search, unlike a rotamer library, relies on exploring all possible combinations of rotatable bonds for all amino-acid residues of a given protein. The search-space is defined by the fine (or coarse) rotation resolution. Several optimization techniques can be used to explore rotations' combinations. Note that in general, a torsion search's rotamers are not pre-generated if used with a fine increment resolution. However, a coarse increment resolution could allow for a pre-generation of all rotamers for a given protein but could also potentially lessen the quality of results. Genetic algorithms, for example [45] [58] [49], were successfully used to search the torsion space of various proteins. A large fraction of a torsion search-space contains useless rotameric states. We speculate that a torsion search has a higher probability of exploring states that could lead the algorithm to permanently fall into local optimum.

4.1.3 MCCE Rotamer Generation

Unlike a fine increment resolution torsion search and a rotamer library, MCCE's generation of rotamers relies on an exhaustive generation of rotamers. One could argue that its method is similar to that of a torsion search, using a coarse increment resolution to pre-generate all required rotamers. This implies that the increment resolution for rotamer generation cannot be as fine as in a torsion search. For example, enumerating all rotamers for a residue with 4 rotatable bonds at a 30° resolution, would produce 12 rotamers per bond and yield a total of $12^4 = 20736$ possible rotameric states.

In MCCE, residues that are surface exposed have their increment resolutions halved. This is done since surface exposed residues are not folded inside the protein, they are exposed to a solvent solution and they are constrained by their neighbouring residues. As a result, their interactions with the rest of the protein are lessened. Reducing their increment resolution improves generation performance and does not greatly impact on the quality of solutions. In addition to this special case handled by MCCE, rotamers that physically favour hydrogen bonds are specially generated to account for electrostatic forces calculated in MCCE step 3. Finally, rotamers of amino-acid residues with high self-energy - that are physically unfeasible - are removed, reducing the number of rotamers for each residue. To calculate the self-energy of a rotamer, we use the AMBER force-field as used in MCCE. For a given rotamer, we consider its torsion energy, its self van der Waals (VDW) energy and its self VDW energy with respect to the backbone of the protein. MCCE keeps the angles (angular energy) and the bonds (bond energy) constant. Therefore, we do not need to calculate the latter two energetic terms. The VDW interaction between any two atoms is calculated using a truncated Lennard-Jones potential (12-6 LJ) function as shown by Equation 4.1.

$$LJ_{pair}(\sigma, r, eps_1, eps_2) = \begin{cases} 0 & r^2 > 100 \\ 999 & r^2 < 1 \\ \sqrt{eps_1 * eps_2} * \left[\left(\frac{\sigma}{r}\right)^{12} - 2 * \left(\frac{\sigma}{r}\right)^{6}\right] & otherwise \end{cases} \quad (4.1)$$

In this equation, the variable is the distance between the two atoms, is the geometric mean of the two atoms' depths of their potential wells as shown in Figure 4.1 and is the arithmetic mean —— of the two atoms' distance at which their potentials are zero. The – term describes the short-range repulsion force while the – term accounts for the long-range attraction forces. We also show the dependence of the potential energy as the distance increases in Figure 4.1.

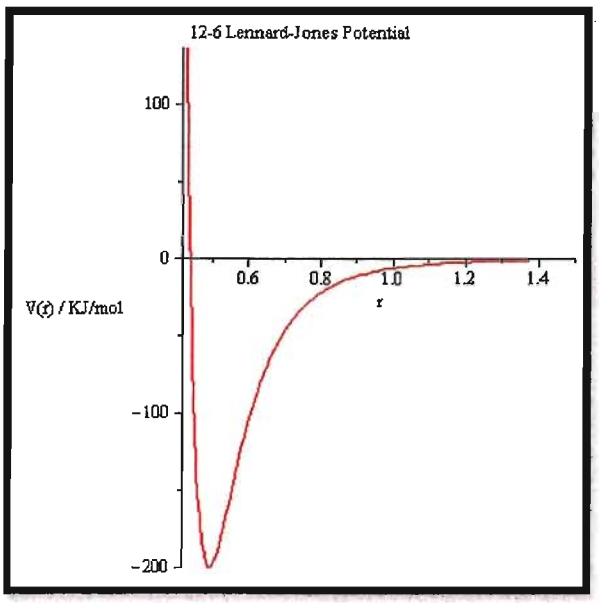

Figure 4.1: 12-6 LJ Potential of the function

– –

The torsion energy of a rotamer is obtained by twisting a rotatable bond. This is calculated using the Amber's force-field torsion energy term as shown by Equation 4.2.

– (4.2)

is the dihedral force constant, is the multiplicity of the function, is the dihedral angle and is the phase shift. Finally, the self-energy of a rotamer is evaluated by Equation 4.3 over all atoms, pair-wise, in Kcal/mol units.

(4.3)

4.2 Problem Encoding and Algorithm Description

We decided to use an integer-coded chromosome representation. The genes of a chromosome, each representing an amino-acid residue, may occupy a value in the range $[1..X]$ where X is the total number of rotamers generated by MCCE for that residue and the occupied value is an integer index mapped to a rotamer in the protein structure. To represent a chromosome, we created an abstract chromosome data structure to store several details of evolution. For any given chromosome (protein conformation), we keep track of its residues' pair-wise energy interactions. This information will be essential for the post-sampling filtering, as we describe in Section 4.3.7. We also monitor whether a chromosome mated, was copied over and if the it needs to have its fitness re-evaluated. Details are shown in Table 4.2 below. To help describe the genetic algorithm and its operators, Figure 4.2 depicts the control flow of our algorithm, beginning with a random initialization of the genetic population. Our algorithm then evolves over several generations until convergence or until the algorithm reaches its maximum allowed number of generations. Each step will be fully described in the following Section 4.3.

Table 4.2: Chromosome data structure.

```
typedef struct {
  float min,vdw,occ_fit;
}INTERACTION;

typedef struct {
  int *genes;
  char flag,mated,copied;
  float fitness;
  INTERACTION *pw_vdw;
}CHROMOSOME;
```

Our hybrid algorithm works with a set of rotamers previously generated by MCCE. From this set, we first determine the total number of residues, N, having more than 1 rotamers and evaluate their static (non-changing) self-energy using Equation 4.3. N also represents the chromosome length of our genetic algorithm. This information is stored in the protein structure for all rotamers of each residue. A rotamer is made up of a list of connected atoms. This list contains heavy and hydrogen atoms. Since we did not use hydrogen atoms in this part of the software, verifying whether an atom was a hydrogen or

a heavy atom was going to be a detriment to our chromosome fitness evaluation function. As such, a new list of heavy atoms, excluding all hydrogen atoms, was pre-built thus avoiding the verification of the atom's type.

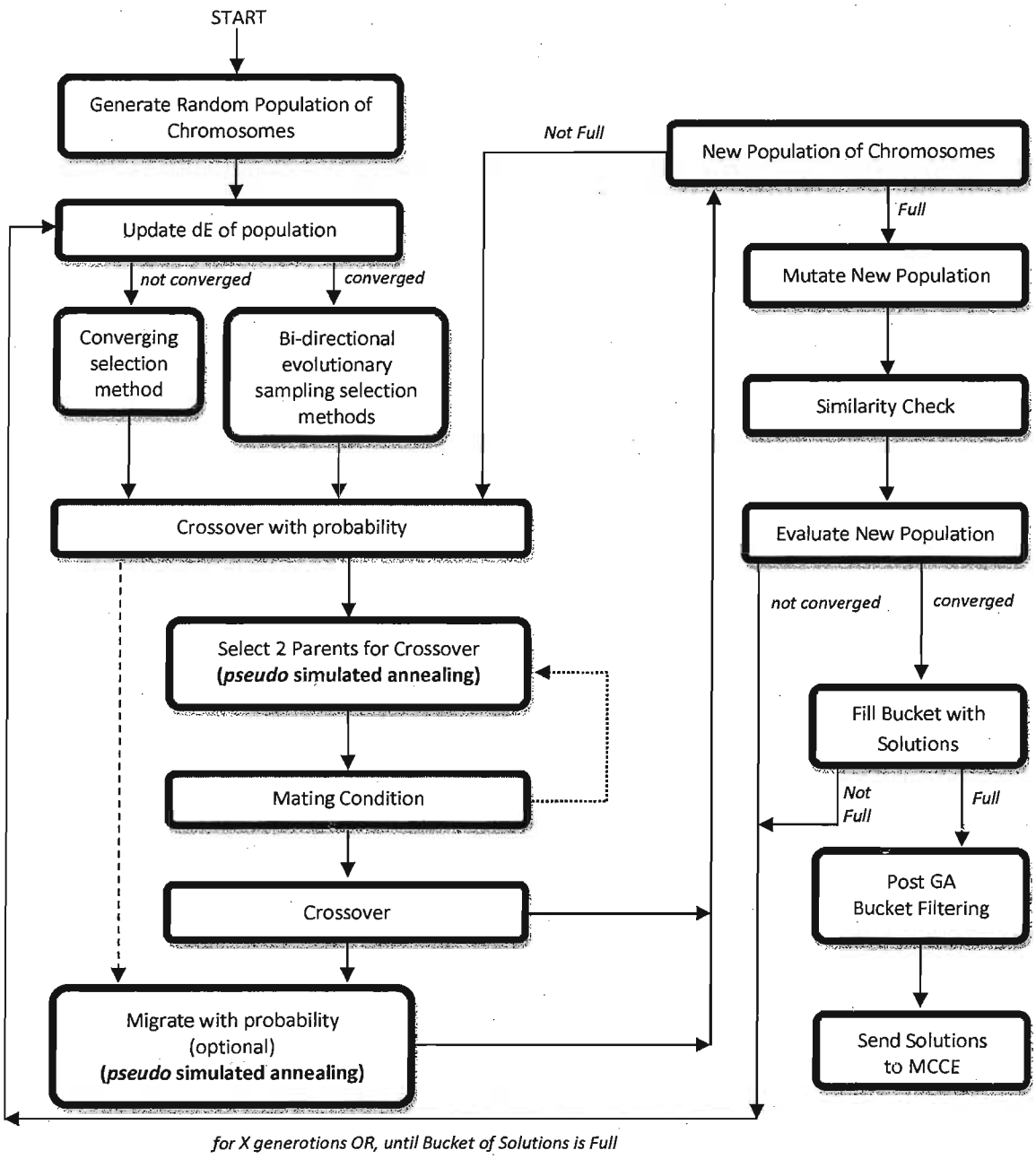

Figure 4.2: Algorithm Flow of Hybrid GA/*pseudo*-SA.

4.3 Genetic Operators

4.3.1 Selection Pressure and Pseudo Simulated Annealing

Heuristic search algorithms are traditionally designed to produce a near-optimal solution. However, our goal is to find a set of rotamers for all the amino-acid residues of a protein that are occupied at a physiological temperature. Temperature adds a kinetic energy term to the total energy of a protein, thus allowing more than one optimal conformation to coexist in the molecular ensemble. We adapted our hybrid GA/*pseudo* SA to our needs so that it did not only converge to a near-optimal solution. Upon convergence of our algorithm, the GA population of rotamers is characterized by an energy distribution similar to that of a molecular ensemble at a certain temperature. This approach allowed us to efficiently sample for occupied rotamers. The sampling technique will be discussed in detail in Section 4.3.6. We will now expand on the theory describing energetic dependence of state occupancy.

In our algorithm, a protein state (or conformation) is referred to as a chromosome and has a fitness value. This fitness value represents the potential energy for this given state. In nature, the probability of a state i with energy E_i to occur is given by the Boltzmann exponential law $e^{-\left(\frac{\Delta E}{kT}\right)}$. The parameter $k = 1.9872065E^{-3}$ (Kcal/mol)/Kelvin is the conversion factor from temperature to energy, T is the temperature in Kelvin units and $\Delta E = E_i - E_o$, where E_o is the optimal low energy. The probability of the state i has a significant contribution if $\Delta E <$ kT. Note that with the Boltzmann constant k, we can compare our chromosome fitness value to temperature and therefore calculate the *pseudo* "temperature" of our system (chromosome population). To help conceptualize all of the above we show these concepts as an example for the staphylococcal nuclease protein (1TR5) in Figure 4.3 below.

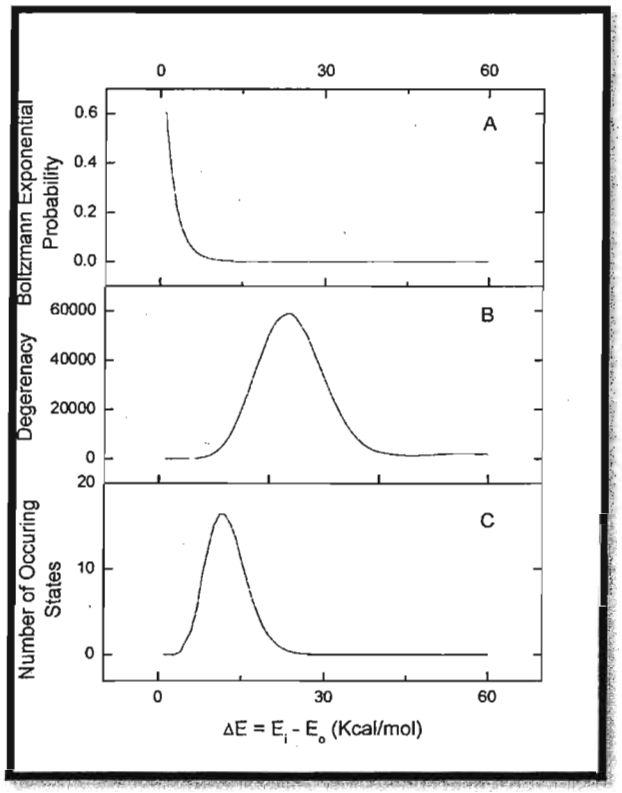

Figure 4.3: (A) Boltzmann Exponential, (B) Degeneracy, (C) Number of Occupied States

Panel A was obtained using — . For this example, which means the temperature Kelvin. We obtained panel B by random uniform sampling of states for only the first 10 residues of the protein 1TR5; it represents the total number of states at each energy interval, called the *degeneracy*. Finally, panel C is the result of multiplying the first two results together, producing the actual number of occupied states based on their probabilities of occurrence. The area under the curve of panel C gives the partition function of the system and indicates the states that are contributing to the total energy. We tried to uniformly sample states for more than 10 residues but because of the factorial nature of the problem, they were impossible to obtain.

For our final selection function operator, we exploited the Boltzmann probability of states existing with energy higher than the minimum, at non-zero temperature. This allowed annealing of our selection function at each generation by continuously reducing the probability of individuals with higher energy being selected, using a control parameter. This parameter is calculated as the average population deviation (dE) from the best

fitness. At each generation of the genetic algorithm its value is updated. The average deviation gives an idea of the current state of the genetic population in terms of convergence towards a near-optimal solution. We expect that chromosomes with very high energy are going to mislead the average deviation and limit our ability to properly anneal our system. Thus, the function takes a percentage of the best chromosomes of the population when evaluating this value. It was empirically selected to be 70%. A pseudo-code of the simple dE calculation algorithm is shown in Table 4.3.

Table 4.3: Population average deviation.

```
Sort Chromosomes in Ascending order of fitness (best to worst);
n <- Select 70% of the best ones;
Sum <- Sum of their Fitness difference to the Best one
dE <- Sum / n;
```

Gradual cooling using this control parameter also avoids pre-mature convergence to a local optimum and encourages high energy states to exchange genetic information between low energy states. In the end, we opted for a merged 2-Tournament selection with the probability of states previously discussed. Table 4.4 shows the pseudo-code of our selection method. The function *random_0_1()* generates a uniformly distributed pseudo random number in the range [0..1) and log_e is the logarithm base e.

Table 4.4: *Pseudo* Simulated Annealing Selection function

```
p1 <- Pick randomly from population
While(no individual has been selected) {
       p2 <- Pick randomly from population;
       IF(Fitness(p2)<Fitness(p1)) Then p1 <- p2;
       roll <- (-dE/kT)*log_e(random_0_1());
       IF(Fitness(p1)-Fitness(Best)<roll) Then Select p1;
}
```

Upon initial execution, the algorithm often selects solutions of high fitness values. As dE of the population gradually lowers, the annealing selection pressure becomes more effective and low energy chromosomes are selected more frequently, analogously to simulated annealing.

4.3.2 Crossover and Mating Rules

We decided to modify a general *N*-Point crossover operator to introduce a high degree of randomness between mated chromosomes. In general, an N-Point crossover uses a fixed number of cutting points (1,2,3...). Genetic material between each pair of cutting points is then exchanged between two parent chromosomes as described in Section 2.1 of Chapter 2. Instead of using this more general approach, we required that at least 1 gene was not swapped between two consecutive pairs of two cut points.

To better conceptualize this idea, we consider a binary chromosome encoding example. First, for a given crossover operation, a bit-mask is generated. Genes are randomly assigned a value of 1 or 0 for a total number of *N* bits. No two consecutive 1s in the bit-mask are exist. There must be at least one gene set to 0 between any two consecutive 1s. This binary example idea is applied to our integer-coded hybrid GA/SA algorithm.

In addition, we also used a random number of cutting points for each crossover operation, versus using a fixed number of cutting points for all crossover operations as used in an N-Point crossover. The number of cutting points was a random value between 1 and 40% of a chromosome's length. We produced two offspring, inserting them one at a time into the new population to ensure that our new population was not overflowing. We use a full generational GA approach. Parent chromosomes may be copied into the new population through the use of the optional migration operator, as discussed in Section 4.3.6. The two parents are not inserted in the new population.

We also decided to use a *mating condition constraint operator*. Any two chromosomes A and B can mate if and only if A and B are different (no asexual reproduction) and at least one of A and B have not already mated. This condition method favours new reproduction cycles between individuals of a population thus contributing to a constant introduction of new chromosomes. It also serves as a more constrained mating pressure amongst individuals, giving the chance for other individuals to be selected for mating. If conditions 1 and 2 are not met, then two new parents are selected.

4.3.3 Fitness Evaluation Function

The fitness function is based on Equation 4.1 and is only performed for heavy atoms; no hydrogen atoms were included since we did not include the electrostatic energy term nor ionization states. The electrostatic term is calculated pair-wise for all rotamers in MCCE step 3. The fitness of a given chromosome C is the sum of the pair-wise interactions of all atoms of the rotamer of a residue against the atoms of another rotamer over all other residues. The fitness of an individual is summed with its self-energy, that was pre-calculated using Equation 4.3. We calculate the fitness of all chromosomes of a population on-the-fly in parallel, using all available CPU cores. If a computing node has enough free RAM, all residues' rotamers' pair-wise interactions are pre-calculated in parallel using all available CPU cores and are then stored in a ragged triangular matrix. Speed up differences between on-the-fly calculations versus pre-calculations and storing in a matrix are available in Table 5.4 of Section 5.1 of Chapter 5. Table 4.5 shows the pseudo-code of the implemented fitness function.

Table 4.5: Evaluation function

```
#Using all available CPU Cores in parallel
For each Chromosome X in the population {
        X.fitness <- equation_4.3();
        For each rotamer I in the chromosome X {
                For each atom K of the rotamer I {
                        For each rotamer starting at J=I+1 in the chromosome X {
                                For each atom P of the rotamer J {
                                        R <- squared_distance(atom[K], atom[P]);
                                        X.fitness   += equation_4.1(atom[K],atom[P],R);
                                        X.pw_vdw[I] += equation_4.1(atom[K],atom[P],R);
                                        X.pw_vdw[J] += equation_4.1(atom[K],atom[P],R);
                                }
                        }
                }
        }
}
```

4.3.4 Elitism

In [45], Milan Keser and Samuel I. Stupp suggested that the use of elitism in their GA would have considerably improved performance. We took this idea into consideration and initially decided to implement a function that would choose a percentage of the best chromosomes at each generation and move them into the new population. When working with large population sizes, thousands and above, a 1% elitism may represent a larger

population percentage than initially wished for. It can thus become detrimental to the algorithm, forcing a pre-mature convergence of solutions. In our case, we simply did not want to lose the best solution. While our aim was to have our algorithm find an acceptable near-optimal solution, we stress that we were uninterested in a premature quickened convergence. We finally opted to keep only one best chromosome at each generation, equivalent to $\frac{1}{population\ size}$ % elitism. The best individual copied over to the new population is not subjected to mutation, though it is able to mate with other chromosomes. To indicate that its fitness does not need to be re-computed, a flag variable is used.

4.3.5 Mutation, Migration and the Similarity Operators

We remind the reader that our algorithm for side-chain packing is to be included in a live distribution version of the MCCE software. The optional migration operator, although not used in our benchmark tests, was implemented for completeness and because MCCE users may require it. This genetic operator copies a selected individual from the population into the new population. The selection is done using the same Boltzmann derived function as the one in Table 4.4 of Section 4.3.1. When copied over, a chromosome's fitness does not change and as such, no re-computation of its fitness is required. To indicate this feature, a flag variable is used. A migrated chromosome still has a chance of mating.

The mutation operator randomly determines if a chromosome should be mutated. If so, then only one gene is selected at random and the gene has its value changed to a random value in the range [1..X], where X is the total number of rotamers for a specific residue. Mutated chromosomes have their copied flags turned off, requiring re-computation of their fitness.

The similarity operator or *niching* operator, was implemented to take care of duplicate chromosomes in a population, as explained in Section 2.1.2. If a chromosome A is genetically identical to chromosomes B, C and D, then the chromosomes B, C and D

have their genes mutated at 100% probability. Our justification for looking at the genetic differences and not their fitness differences comes from the fact that a chromosome A could differ in one gene from chromosome B but have quite dissimilar fitness. This operator is extremely important to avoid premature convergence and prevents dE from reaching 0 (as introduced in Section 4.3.1), by constantly inserting new random genetic material in the population. Small values of dE such as 0.5 would indicate a good convergence while retaining diversity. Values near 0.0 show that solutions are too similar to each other in terms of fitness even although they could be genetically constructed in varying ways. Genetically similar solutions will be sampled for as described in Section 4.3.6, after convergence is attained. This operator forbids dE to reach a value near 0 by continuously enforcing genetic diversity while achieving adequate convergence. In the research conducted by Milan Keser and Samuel I. Stupp [45], they employed a relaxed version of this similarity operator. Their *niching* operator looked at a percentage of genetic dissimilarity between duplicate chromosomes. Their function mutated each gene at a much lower probability rate than ours. We are unsure which of these two methods is better, though their technique implied two extra control parameters which introduced more levels of randomness in the evolution process.

4.3.6 Dynamic Bi-Directional Evolutionary Sampler

In his book *Wonderful Life* [59], Stephen Jay Gould discussed the concept of "rewinding" the universe to let evolution begin its cycle again. This notion has created controversy among evolutionary philosophers since its time of publication. Furthermore, he stipulated that when conditions of evolution change rapidly, the survival of a species depends more on chance and features beneficial under future conditions than features best adapted under current constraints. Using his suggested idea, we developed a *dynamic bi-directional evolutionary sampler* that behaves similarly to reverse evolution. It is a novel and simple technique designed to generate a statistical distribution of solutions in the vicinity of a near-optimal solution, by allowing genetic materials to undergo continuous backward and forward evolution with fast changing conditions and continuous alternation of selection pressure. The point at which to sample can be explicitly specified and therefore does not require near-optimal solution convergence prior to its use. Our

sampler offers a "localized" sampling around a specific location in the search-space and allows for sampling of solutions which could not have been otherwise obtained by following a regular converging evolutionary algorithm. Dynamic bi-directional evolutionary sampling also strongly enforces diversity among chromosomes. This technique is closely coupled with two modified versions of the previously discussed selection method of Table 4.4 in Section 4.3.1. For the purpose of side-chain packing, the evolutionary sampling is engaged only after having achieved an adequate convergence through a normal GA/SA execution. The reader may wish to refer to Figure 4.2 of Section 4.2 as a reminder of the control flow of the algorithm.

We recall that our goal is to obtain a set of different rotamer states at a certain temperature. To represent a specific location for the dynamic bi-directional evolutionary sampling, we introduced a new parameter, the *distribution centre*. The question of where to set the center of distribution for optimal sampling is open, although the results of Chapter 5 indicate that the general rule of thumb is to sample in the vicinity of the near-optimal solution. We also define two new Boltzmann derived selection functions: one for converging and one for diverging, as shown in Tables 4.6 and 4.7. These two selection functions are solely used for the purpose of the dynamic bi-directional evolutionary sampler. Unlike the original selection function definition in Table 4.4 of Section 4.3.1, the converging and diverging selection functions do not use a merged 2-Tournament selection pressure. Instead, we used a merged random selection pressure.

Table 4.6: Converging selection function for the dynamic bi-directional evolutionary sampling

```
While(no individual has been selected) {
       p1 <- Pick randomly from population;
       IF(Fitness(p1)-Fitness(Best)<distribution centre) Then Select p1;
       roll<- (-dE/kT)*log_e(random_0_1());
       IF(Fitness(p1)-Fitness(Best)<roll) Then Select p1;
}
```

Table 4.7: Diverging selection function for the dynamic bi-directional evolutionary sampling

```
While(no individual has been selected) {
       p1 <- Pick randomly from population;
       IF(Fitness(p1)-Fitness(Best)>distribution centre) Then Select p1;
       roll<- (-dE/kT)*log_e(random_0_1());
       IF(Fitness(p1)-Fitness(Best)>roll) Then Select p1;
}
```

Individuals for which their fitness difference to the best individual falls below the distribution center, are randomly selected. Figure 4.4 enhances the differences in including a merged random selection as opposed to not including it. One can observe large and unpredictable fluctuations without the merged random selection pressure. Selective pressure is only applied on individuals for which their fitness difference to the best individual falls above or below the distribution centre, depending on which of the two selection functions is used at that time.

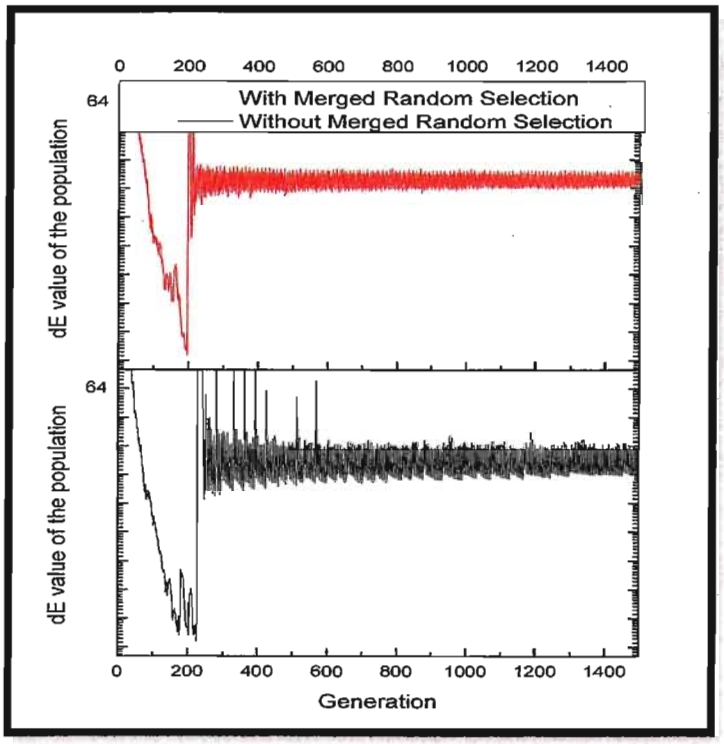

Figure 4.4: Effect of merged random selection for the Evolutionary Sampler
The first ~200 generations search for the near-optimal solution

The point in time at which the diverging function is used is based on the dE value of the chromosome population. If then the dynamic bi-directional evolutionary sampler switches to the diverging function until . When this happens, the evolutionary sampler switches back to the converging function for the following generations until . This switching is repeated until we have filled a bucket with enough chromosome solutions. During the initial generations of the evolutionary

sampler we observed large fluctuations in the dE values. Quickly thereafter, we suspect that dE becomes cyclic and even upper and lower bounded, slowly converging to the actual specified distribution centre.

In order to correctly and properly assess the validity of our dynamic bi-directional evolutionary sampler, we need mathematical proofs to elaborate on a sampler's distribution convergence rate. To do this requires theories of Adaptive Monte-Carlo Markov-Chain (MCMC) distribution convergence which are today still being expanded upon. Several studies have indeed been dedicated to the understanding of this type of process and it is without a doubt, a very difficult problem to solve [60] [61]. In general, MCMC algorithms are employed in Computer Science, Physics and many other fields of science requiring statistical sampling from complex high-dimensional probability functions. Ideally, we would want to show the rate at which our dynamic bi-directional evolutionary sampler converges to our target statistical distribution.

Due to the complexity in proving the latter, we illustrate for now the observed statistical phenomena in Figures 4.5, 4.6 and 4.7. Figure 4.5 describes the cyclic dE behaviour of the dynamic bi-directional evolutionary sampling process for the lysozyme protein (4LZT), filling a sampling bucket with 3,300,000 solutions. We identified this cyclic behaviour to be similarly well defined for our 24 benchmarked proteins.

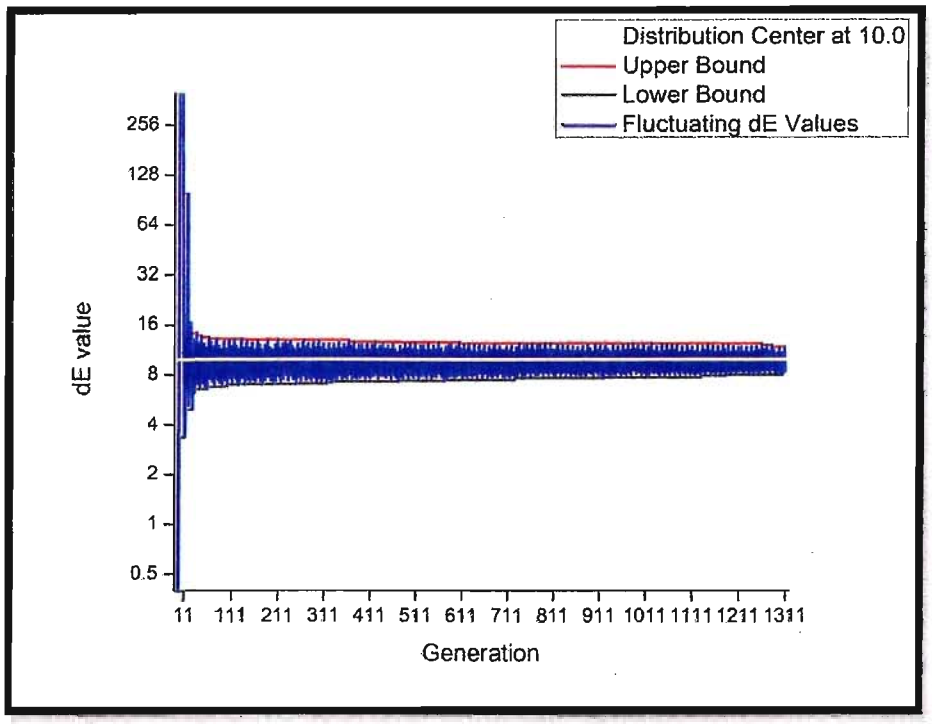

Figure 4.5: Bounded Dynamic Bi-directional Evolutionary Sampling and Cyclic behaviour over 1311 generations.

Figures 4.6 and 4.7 illustrate the density of the bucket as it gets filled at two different generation points in time for the egg white lysozyme (4LZT) protein. We can observe that under the criteria of the converging and diverging selection techniques, we obtained a concentrated distribution of solutions around the distribution centre. The interesting point is that this technique can reproduce similar statistical distributions at virtually any centered position from a near-optimal solution. Figures 4.6 and 4.7 were obtained by sampling each GA population generated at each generation during which the dynamic bi-directional evolutionary sampling was engaged. When a chromosome's fitness difference to the best fell within the sampling range of *2 epsilon times the distribution center* (2*10.0=20.0Kcal/mol units), we included it in our bucket and allowed duplicates. After several hundred generations of filling of the sampling bucket, the rate at which new unique rotamer states were found was considerably reduced. It is also possible to find a better near-optimal solution while performing the dynamic bi-directional evolutionary sampling. Although the probability of this happening is fairly low, we observed this phenomenon several times.

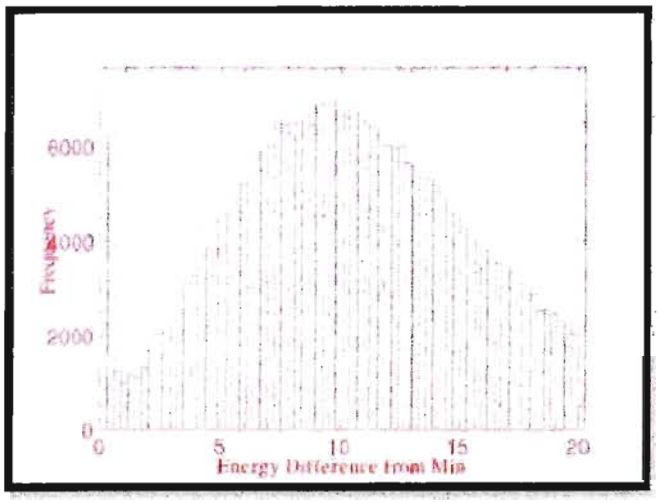

Figure 4.6: Sampling density of 300,000 solutions, distribution centre at 10.0. Including solution fitness differences up to 20.0kcal/mol (2eps*10.0).

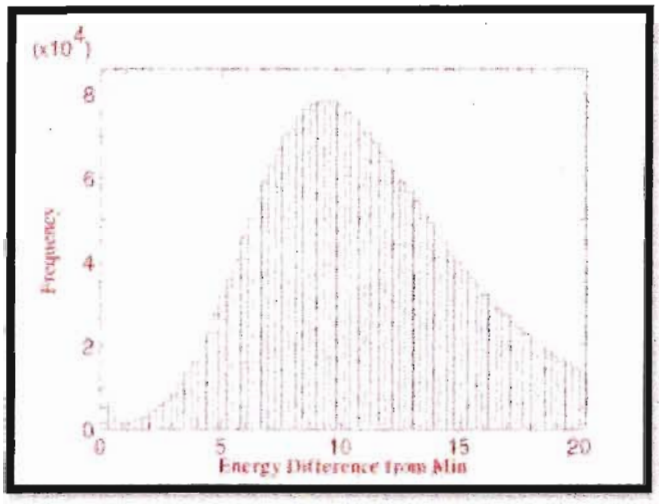

Figure 4.7: Sampling density of 2,700,000 solutions, distribution centre at 10.0. Including solution fitness differences up to 20.0kcal/mol (2eps*10.0).

The key difference between Figures 4.6 and 4.7 is the shape of the distribution, taken at two different points in time. Visualizing this information also gives us a preliminary indication of the rate at which our dynamic bi-directional evolutionary sampler may converge to the target statistical distribution. In theory, it is possible to find the *real* location of the distribution center. One would have to randomly sample all solutions starting from the near-optimal solution. Sampling could be done until the observed number of states start to decrease. Since the size of proteins we generally encounter is of

40+ amino-acid residues, and due to the factorial nature of the problem, this is not yet a realistic approach.

4.3.7 Post-Sampling Filtering

As described in the previous Section 4.3.6, we sampled solutions by looking at chromosomes' fitness, where each represented a possible protein conformation. It is important to also locally filter each residue's rotameric states due to our energy function's missing electrostatic forces. To compensate for this, while filling the bucket we keep all of a residue's rotamers falling within *X* Kcal/mol units from the minimum energy the residue has ever occupied, and remove the others. The filtering looks at the residues' pair-wise VDW interactions, including self-energy. It is worth noting that a local residue filtering of 3Kcal/mol on a given protein may not produce as good results if performed on different protein. Our results section in Chapter 5 will show this fact in more detail.

In addition to this filtering, we implemented a filtering based on rotamers' occupancy. This method looks at the number of times a certain rotameric state occurs in the sampling bucket for a specific residue. Rotamers of each residue that fall below this threshold parameter are removed. We set this occupancy cut-off as an input parameter and it can be turned off using a value of 0.0. We also generate three types of histograms. One is generated for the final GA population that emerges after performing a dynamic bi-directional evolutionary sampling. A second one is generated for the sampling bucket screening and represents the statistical distribution of chromosomes' fitness differences to the best chromosome's fitness. Finally, we generate histograms for each residue, showing the statistical distribution of rotameric states at the residue level. Rotamers are then sent back to MCCE for calculation of electrostatics in step 3 and Monte-Carlo sampling in step 4 for residues' pKa prediction.

4.4 Algorithm Control Parameters

Since some details of our algorithm can be easily forgotten due to its level of complexity and involvement, we describe all of the parameters that are used to control our evolutionary algorithm. For a complete list of parameters used by both MCCE and our algorithm, refer to Appendix C.

Population size: The size of the pool of chromosomes at each generation. Note that two such pools are required, one for the current generation and one used for filling the next generation.

Mutation rate: The rate at which individuals should be mutated. Usually a low rate is used, 0.1 and below.

Migration rate: The rate at which individuals are moved from the current generation into the new generation population. Implemented as an optional operator for the benefit of MCCE users.

Crossover rate: Controls the probability at which two individuals mate.

Maximum cut points: The maximum number of cutting points to use the crossover operator. If this value is larger than 40% of the length of a chromosome, the default value of 40% of the chromosome length is applied.

Seed value: Parameter for the random number generator. A value of -1 indicates the use of a random seed value obtained by calling the function $time(0)$.

Generations: The maximum number of generations for the genetic algorithm before which engaging the dynamic bi-directional evolutionary sampler, regardless of the convergence criterion.

Distribution Centre: The parameter used by the dynamic bi-directional evolutionary sampling process, in Kcal/mol units.

Distribution epsilon: The epsilon parameter to control the range of solutions to look at when filling the sampling bucket. We recommend values between 0.5 and 2.0.

Convergence epsilon: Controls the convergence criterion of the algorithm, at which point dynamic bi-directional evolutionary sampling is started if met.

Bucket Size: The maximum number of chromosome solutions to insert in a sampling bucket.

Local Residue Cut-off: The energy, in Kcal/mol units, for local residue filtering of rotamers in the post-GA filtering.

Occupancy Cut-off: The percentage below which occupied rotamers will be removed in the post-GA filtering. A value of 0 disables this filtering.

Chapter 5

Results Analysis and Conclusion

In this final chapter, we present benchmark results applied to several proteins. We compare our results against experimental data and MCCE's published results and compare total execution time of our algorithm versus MCCE's algorithm. Comparison of our hybrid search technique against a vanilla genetic algorithm is also performed using Student T-tests. We enhance the issues of sampling at different distribution centers for the evolutionary sampler and show that post-filtering methods of 3Kcal/mol and 5Kcal/mol have variable outcomes for different proteins.

5.1 Comparison to MCCE and Experimental Results

To provide consistent analysis with respect to the performance of MCCE, we decided to make use of the 2009 published benchmarks of MCCE [22]. Twenty-four proteins that were used are shown in Table 5.1, with the total number of residues defining the partial problem size of our genetic hybrid algorithm.

Table 5.1: Protein data set

PDB Code	Number of Residues in Chromosome	PDB Code	Number of Residues in Chromosome
1PGB	52	1GOA	141
1LE2	138	1RGG	88
1NFN	127	1I0V	92
1GS9	137	1DG9	151
1IGD	57	1H4G	182
4PTI	52	1XNB	182
1IG5	70	1KXI	60
4LZT	117	2CI2	62
1DWR	141	2CPL	141
1A6K	143	1HNG	164
3RN3	121	3EBX	57
1BEO	95	1PPF	52

Due to our dependence on MCCE for steps 3 and 4, we decided to perform six different sampling tests for each protein. We ran these benchmarks, as shown in Table 5.2, to investigate whether the post-sampling filtering methods, the bucket sampling ranges and including more or fewer rotamers were improving or worsening pKa prediction results. Each of the six tests were performed only once due to the large amount of computational time required. Tests were conducted using the same initial seed value for all proteins. It was also necessary to re-run regular MCCE on our machines to reproduce their published results. All of the simulations were executed on Intel Core i7 920 @2.67Ghz machines. Subjected to six different tests, each protein's solutions were extracted from a sampling bucket, generated using the dynamic bi-directional evolutionary sampler. Due to the outsized number of input parameters for both MCCE and our side-chain packing algorithm, we made them available in Appendix C.

Table 5.2: Benchmarking tests

Tests Performed	Sampling Ranges	Post Sampling Filtering
#1 Regular MCCE2 (12 rotations/bond)	-	-
#2 Regular MCCE2 (6 rotations/bond)	-	-
#3 GA/SA (12 rotations/bond)	2eps*10=**20kcal/mol**	0.1 occupancy cut-off **3kcal/mol** local residue
#4 GA/SA (12 rotations/bond)	2eps*15=**30kcal/mol**	0.1 occupancy cut-off **3kcal/mol** local residue
#5 GA/SA (12 rotations/bond)	2eps*10=**20kcal/mol**	0.05 occupancy cut-off **3kcal/mol** local residue
#6 GA/SA (12 rotations/bond)	2eps*15=**30kcal/mol**	0.05 occupancy cut-off **3kcal/mol** local residue
#7 GA/SA (12 rotations/bond)	2eps*15=**30kcal/mol**	0 occupancy cut-off **3kcal/mol** local residue
#8 GA/SA (12 rotations/bond)	2eps*15=**30kcal/mol**	0 occupancy cut-off **5kcal/mol** local residue

To illustrate the difference in dynamic bi-directional evolutionary sampling at different distribution centres, we superimposed two histograms generated for 2 million chromosomes for the protein 4LZT in Figure 5.1.

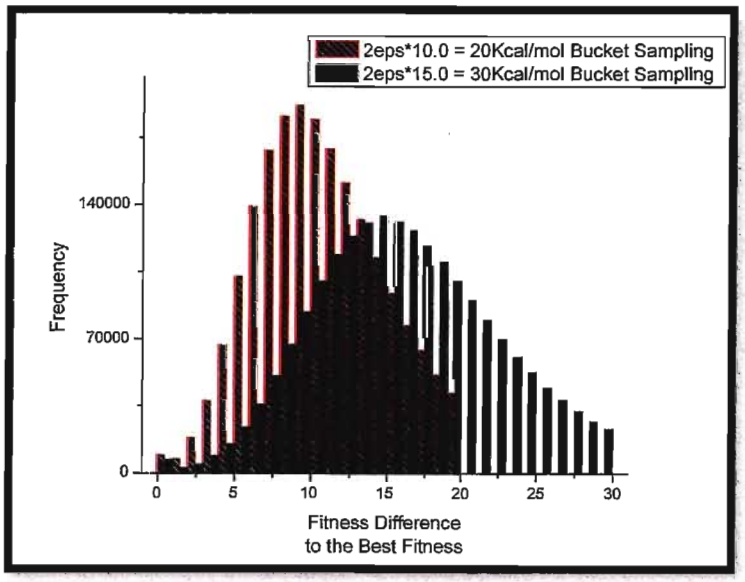

Figure 5.1: Overlap of two different sampling distributions
Two different distribution centres: 10.0 and 15.0

The overlapping and non-overlapping regions are made very clear. Both sampling distributions are centered around their respective 10.0 and 15.0 values. With this image, we can better perceive the different regions of the search-space that are sampled via two different sampling ranges.

Several trends worth noting emerged from our benchmark result. For the following results, the protein 3EBX was excluded due to its awry results for both MCCE and our algorithm. First, for 13 out of 23 proteins, at least 1 out of 6 of the sampling tests using our hybrid search technique outperformed MCCE for both its published results and regular runs of MCCE. For the other 10 out of 23 proteins, our algorithm's performance nearly matched that of MCCE. For each test performed on a protein, we calculated the root mean square deviation (R.M.S.D.) from the experimental pKa, the average difference from experimental pKa to see any systematic shifts, the pKa difference standard deviation to see how consistent the predictions were and the sum of errors. An error occurred when the pKa difference for a residue to the experimental pKa was greater or equal to 1.0. For a more representative spread of errors, we accumulated these errors at different intervals. Residues that MCCE failed to predict were accumulated as "out of range". For each benchmark test, all of these statistics were individually calculated and also averaged over all 23 proteins. We recommend referring to Appendix B for the full detailed summary of each protein's tests.

For simplicity, we first show in Table 5.3 the pKa R.M.S.D of each benchmark test averaged over all 23 proteins. The best averaged pKA R.M.S.D. are high-lighted in bold green. A small R.M.S.D is better than a large value. Based on these results, we decided to show the complete results of our worst and best GA benchmarks, tests #3 and #8 respectively. Figure 5.2 shows the predicted pKa's for the first 12 proteins and Figure 5.3 shows the remaining 11 proteins.

Table 5.3: Averaged pKa R.M.S.D. over 23 proteins

	Published MCCE	#1 MCCE 12 rot./bond	#2 MCCE 6 rot./bond	#3 GA/SA 20kcal 0.1 occ. 3kcal res.	#4 GA/SA 30kcal 0.1 occ. 3kcal res.	#5 GA/SA 20kcal 0.05 occ. 3kcal res.	#6 GA/SA 30kcal 0.05 occ. 3kcal res.	#7 GA/SA 30kcal 0 occ. 3kcal res.	#8 GA/SA 30kcal 0 occ. 5kcal res.
Averaged pKa R.M.S.D.	**0.940319**	1.015535	**0.97543**	1.018616	0.993114	1.005661	0.990208	1.003899	**0.970789**

Test #8, without occupancy filtering and 5Kcal/mol local residue filtering, proved to be our best method on average. Note that our calculated benchmark results *may* be misleading yet represent an overall *approximate* performance. In reality, we would need a minimum of 30 different random runs for each protein to come to more statistically justifiable results.

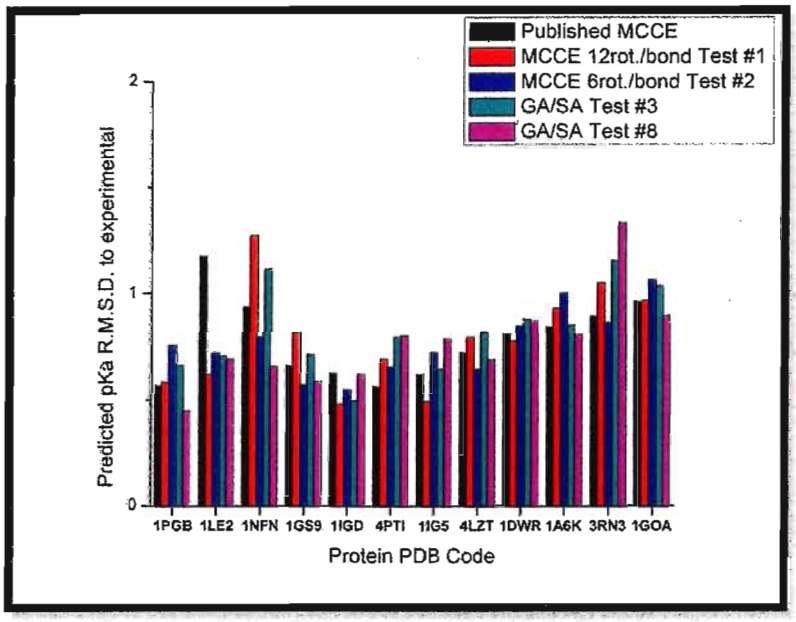

Figure 5.2: pKa R.M.S.D of 12/23 proteins

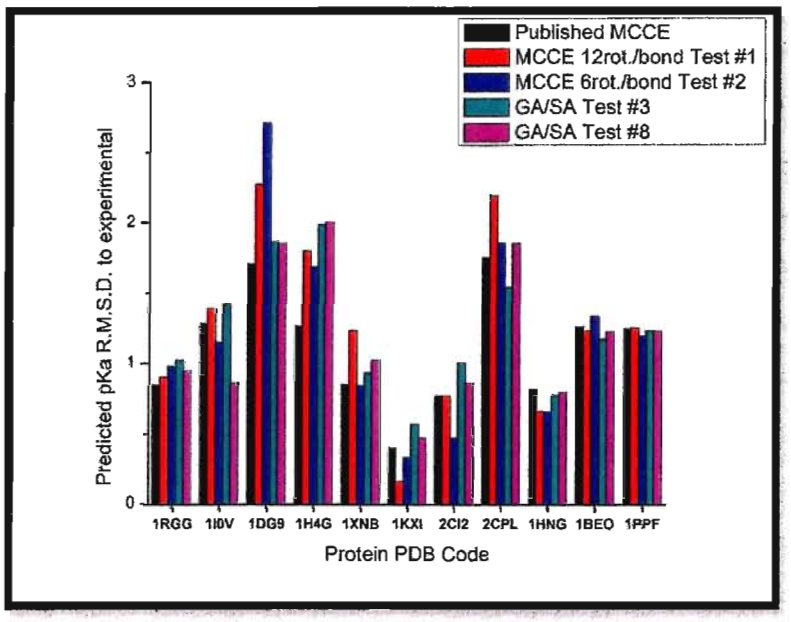

Figure 5.3: pKa R.M.S.D of 11/23 proteins

Secondly, we noticed inconsistencies with MCCE's Monte-Carlo sampling for predicting residues' pKa. We performed a Monte-Carlo sampling (MCCE step 4) independently 30 times and then averaged out the results. These results indicate that in many cases, giving MCCE more rotamers did not improve pKa predictions. We first show that giving MCCE more rotamers produced by our hybrid algorithm did not improve results, as depicted Figure 5.4 for the 1IG5 protein. On the other hand, tests #1 and #2 for regular MCCE runs, produced the expected trend; the more states, the better R.M.S.D.. In fact, the Monte-Carlo sampling of MCCE on which we have to rely for pKa predictions is not a well-converged algorithm. This was confirmed in two private communications (Dr. Marilyn Gunner and Yifan Song, 2010).

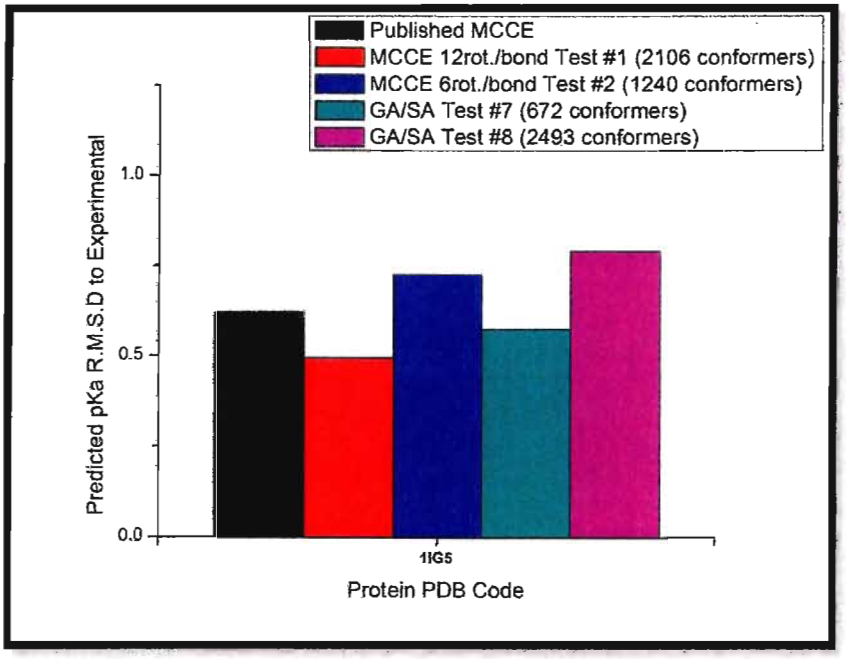

Figure 5.4: Inconsistent trend of protein 1IG5

The published results were outperformed by both our benchmark test #7 and regular MCCE run test #1. The observed trend for tests #7 and #8 also arose several times for regular MCCE run tests #1 and #2. Finally, we show the latter in Figure 5.5 by looking at results of a different protein, 1GS9. Regular MCCE runs with more rotamers (12 rotations/bond) worsened results.

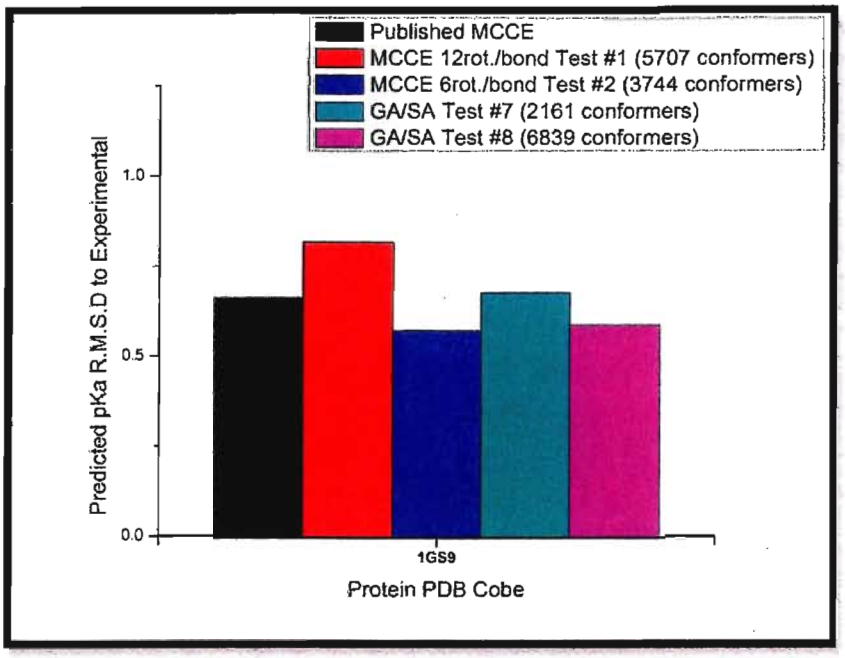

Figure 5.5: Inconsistent trend of protein 1GS9

Test #1 (more states) produced worse results than test #2 (fewer states) for 1GS9. Tests #7 and #8 performed as expected; the more states there are, the better are the results. The published results were also outperformed by regular MCCE test #2 and also by our method test #8.

We suspect that the problem of side-chain packing for residues' pKa prediction requires both a near-optimal conformation of the protein and also a proper sampling of the search-space in the vicinity of this conformation. Complementary to our benchmark results, we offer the calculated speed up of our method versus MCCE's side-chain packing in Table 5.4.

Table 5.4: Speed up of Hybrid Algorithm for side-chain packing

		Without pre-calculation & storing of pair-wise conformers' VDW Interactions		With pre-calculation & storing of pair-wise conformers' VDW interactions	
PDB	Problem Size	GA Speed up vs. MCCE 6 rot./bond	GA Speed up vs. MCCE 12 rot./bond	GA Speed up vs. MCCE 6rot./bond	GA Speed up vs. MCCE 12 rot./bond
1PGB	52	419s/204s = 2.1	1221s/204s = 5.9	419s/34s = 12.3	1221s/34s = 35.4
1LE2	138	3317s/1233s = 2.6	15361s/1233s = 12.5	3317s/122s = 27.1	15361s/122s = 125.5
1NFN	127	2877s/746s = 3.8	9963s/756s = 13.4	2877s/76s = 37.8	9963s/76s = 131.1
1GS9	137	3794s/649s = 5.8	12884s/649s = 19.9	3794s/91s = 41.3	12884s/91s = 140.3
1IGD	57	476s/163s = 2.9	1838s/163s = 11.3	476s/14s = 34.3	1838s/14s = 132.5
4PTI	52	480s/290s = 1.7	1365s/290s = 4.7	480s/29s = 16.4	1365s/29s = 46.6
1IG5	70	655s/305s = 2.1	2096s/305s = 6.9	655s/16s = 40.3	2096s/16s = 128.9
4LZT	117	1409s/355s = 19.7	4483s/355s = 62.8	1409s/42s = 33.1	4483s/42s = 105.3
1DWR	141	3441s/673s = 5.1	8034s/673s = 11.9	3441s/82s = 41.9	8034s/82s = 98.0
1A6K	143	3758s/736s = 5.1	9101s/736s = 12.4	3758s/70s = 53.2	9101s/70s = 128.7
3RN3	121	989s/303s = 3.2	3494s/303s = 11.5	989s/31s = 31.9	3494s/31s = 110.5
1GOA	141	2601s/653s = 3.9	6300s/653s = 9.7	2601s/67s = 38.6	6300s/67s = 93.5
1RGG	88	1891s/167s = 11.3	3646s/167s = 21.8	1891s/18s = 103.2	3646s/18s = 198.9
1I0V	92	496s/205s = 2.4	1807s/204s = 8.8	496s/20s = 24.3	1807s/20s = 88.5
1DG9	151	2414s/635s = 3.8	6406s/635s = 10.1	2414s/68s = 35.3	6406s/68s = 93.7
1H4G	182	2829s/1036s = 2.7	9699s/1036s = 9.4	2829s/99s = 28.5	9699s/99s = 97.6
1XNB	182	1640s/685s = 2.3	6227s/685s = 9.1	1640s/71s = 22.9	6227s/71s = 87.0
1KXI	60	369s/221s = 1.6	3750s/221s = 16.9	369s/40s = 9.1	3750s/40s = 92.0
2CI2	62	1381s/263s = 5.2	6197s/263s = 23.5	1381s/49s = 27.8	6197s/49s = 124.8
2CPL	141	1235s/978s = 1.2	5431s/978s = 5.5	1235s/105s = 11.7	5431s/105s = 51.2
1HNG	164	3100s/1239s = 2.5	19637s/1239s = 15.8	3100s/184s = 16.8	19637s/184s = 106.3
3EBX	57	327s/230s = 1.4	1801s/230s = 7.8	327s/30s = 10.7	1801s/30s = 59.0
1BEO	95	204s/408s = 0.5	890s/408s = 2.2	204s/54s = 3.8	890s/54s = 16.5
1PPF	52	568s/188s = 3.0	1053s/188s = 5.6	568s/29s = 19.4	1053s/29s = 36.0
Average Speed up		3.4	11.2	30.1	96.9

The general trend is that with more rotamers, the larger is the speed up of our method over MCCE's algorithm. Given enough free RAM on the computing node, pre-calculating pair-wise rotamers' VDW interactions and storing them in a ragged triangular matrix is crucial for optimal speed up.

5.2 Further Analysis

We further analyze the results of our hybrid algorithm versus that of a vanilla genetic algorithm. We selected 3 proteins based on their problem size; 4PTI with 52 residues, 4LZT with 117 residues and 1XNB with 182 residues. In contrast with Section 5.1 tests, for this section we selected only two sampling tests we knew we could execute at a minimum of 30 times. We picked test #7; 2eps*15Kcal/mol bucket sampling, no occupancy cut-off and 3Kcal/mol local residue filtering. We also picked test #8; 2eps*15Kcal/mol bucket sampling, no occupancy cut-off and 5Kcal/mol local residue filtering. Test #7 was selected based on its average performance shown in Table 5.3, producing the smallest R.M.S.D. out of all other GA benchmark tests. For ease of results analysis we also included the other test in its category without occupancy filtering, test #8. We executed our tests randomly 30 times for both the hybrid and the vanilla methods.

5.2.1 Vanilla Genetic Algorithm Operators

The flow of the algorithm is the same as that of our hybrid approach except for a few operators. We had to use our dynamic bi-directional evolutionary sampler to produce a bucket of solutions. Our results are therefore *strongly biased* towards the evolutionary sampler.

Crossover Operator: The simplest 2-Point crossover operator was implemented, as described in Section 2.1.2 of Chapter 2.

Crossover Condition Operator: We also decided to use a mating condition constraint operator as described in Section 4.3.2 of Chapter 4.

Mutation Operator: We used the same operator as described in Section 4.3.5 of Chapter 4. We have omitted the use of the s*imilarity check (niching) operator.*

Selection Operator: A standard 3-Tournament selection function was implemented, as described in Section 2.1.2 of Chapter 2.

Elitism: We decided to keep elitism for the same reason described in Section 4.3.4 of Chapter 4.

5.2.2 Hybrid and Vanilla Genetic Algorithm Results

The statistics shown in Table 5.5 are interesting to study. Numbers highlighted in green show the method for which the results are superior. Similarly, statistics in red show the method for which outcomes are worse. Assuming normal distribution of our data, we also performed a 1-tail unpaired equal variance Student t-test on the 30 best fitness and MCCE step 4 R.M.S.D. values obtained for both methods, for all 3 proteins. We tested the null hypothesis $u_2 = u_1$ against the alternative hypothesis $u_2 - u_1 > 0$ at the 0.05 level of significance. We rejected the null hypothesis if our P-value was < 0.05.

Table 5.5: GA Test #7 Hybrid and Vanilla genetic algorithm results of 3 proteins

	GA Test #7: 2eps*15Kcal/mol *Bucket Sampling* **0.0** *Occupancy Filtering*, **3Kcal/mol** *Local Residue Filtering*					
Statistics	**4PTI Hybrid**	**4PTI Vanilla**	**4LZT Hybrid**	**4LZT Vanilla**	**1XNB Hybrid**	**1XNB Vanilla**
Number of Residues	52	52	117	117	182	182
Number of Runs	30	30	30	30	30	30
Average *near-optimal*	2017.67	2018.08	2898.47	2899.11	-640.59	-640.41
The Best *near-optimal*	2017.04	2017.04	2896.35	2896.75	-646.36	-645.95
The Worst *near-optimal*	2024.74	2020.36	2903.12	2902.81	-636.57	-635.7
***near-optimal* Std. Dev.**	1.93	0.92	1.74	1.27	2.46582	2.09458
Average R.M.S.D.	0.8478	0.8504	0.8081	0.7612	0.97827	1.0153
Best R.M.S.D.	0.7925	0.7781	0.6524	0.6247	0.63425	0.7184
Worst R.M.S.D.	0.9313	0.9059	0.9495	0.8646	1.3313	1.3599
Published MCCE R.M.S.D	0.56088		0.72376		0.85586	
MCCE 12rot./bond	0.69213		0.79606		1.23404	
MCCE 6rot./bond	0.65603		0.64532		0.84292	
Sum of Errors 1-1.5	96	90	64	50	46	44
Sum of Errors 1.5-2.0	30	30	17	22	18	19
Sum of Errors 2.0-3.0	0	0	9	5	13	15
Sum of Errors > 3.0	0	0	0	0	0	0
Average Number Conformers	808.2667	922.5667	1530.03	1656.3	1580.3	1660.6
Average Rotamer Generation Time	1768 s	1768 s	12543 s	12543 s	15040 s	15040 s
Average Step 3 Run Time	1106.6 s	1338.733 s	3158.3 s	3607.97 s	3942.9 s	4270.2 s

Average Step 4 Run Time	113.4667 s	126.3333 s	537.7 s	627.3 s	1399.2 s	1545.3 s
Average Total Run Time	**3191.1942 s**	**2018.079 s**	**16631.79 s**	**17056.91 s**	**21074.5 s**	**21467.2 s**
1-Tail Homoscedastic T-Test of 30 Best Fitness Values	**0.1523**		**0.0546**		**0.3789**	
1-Tail Homoscedastic T-Test of 30 R.M.S.D Values	**0.3842**		**0.0053**		**0.1846**	

The t-tests of fitness for all 3 proteins were inconclusive. The t-test of fitness for 4LZT suggested that with more samples we could obtain statistical evidence that the hybrid method is better than the vanilla method. The t-test of R.M.S.D. values for 4LZT is the only one that showed statistical evidence that, MCCE step 4 with the rotamers we sampled using the evolutionary sampler, outperformed the vanilla method.

On average, the hybrid algorithm found a better *near-optimal* solution more often than the vanilla did. For the proteins 4PTI and 1XNB, the hybrid method produced an average RMSD better than the vanilla by 0.31% and 1.41% respectively. As for 4LZT, the vanilla algorithm produced a better average RMSD by 5.8%. If we look at the sum of errors obtained, the vanilla algorithm was the clear winner. On average, the vanilla algorithm generated a little less than a hundred conformers more than the hybrid method. One may jump to the quick conclusion that having more conformers therefore produced better R.M.S.D. results but it was not the case. To provide evidence of this phenomenon, we show results of the best and worst run based on R.M.S.D values for all 3 proteins in Table 5.6. On a different note, the dE (Section 4.3.1) of the vanilla algorithm reached a value of 0.0 several times while executing the evolutionary sampler. This crashed the algorithm and required a re-start of the simulations. This effect led us to believe that the vanilla algorithm had a poor diversity level.

Table 5.6: Best and Worst Runs for 4PTI, 4LZT and 1XNB

		GA Test #7: *2eps*15Kcal/mol Bucket Sampling* *0.0 Occupancy Filtering*, **3Kcal/mol** *Local Residue Filtering*					
	Statistics	**4PTI Hybrid**	**4PTI Vanilla**	**4LZT Hybrid**	**4LZT Vanilla**	**1XNB Hybrid**	**1XNB Vanilla**
	Number of Residues	52	52	117	117	182	182
Best	**Nb. Conformers**	**786**	**884**	**1441**	**1643**	**1671**	**1732**
	Fitness	2017.03	2018.81	2902.76	2899.41	-636.82	-638.05
	R.M.S.D.	**0.7925**	**0.7781**	**0.6524**	**0.6247**	**0.6342**	**0.7183**
	Sum of Errors 1-1.5	2	2	1	1	1	2
	Sum of Errors 1.5-2.0	1	1	0	0	0	0
	Sum of Errors 2.0-3.0	0	0	0	0	0	0
	Sum of Errors > 3.0	0	0	0	0	0	0
Worst	**Nb. Conformers**	**679**	**1040**	**1557**	**1791**	**1592**	**1624**
	Fitness	2017.03	2018.05	2897.64	2899.79	-641.58	-644.82
	R.M.S.D.	**0.9313**	**0.9059**	**0.9495**	**0.8646**	**1.3313**	**1.3598**
	Sum of Errors 1-1.5	4	4	4	0	2	1
	Sum of Errors 1.5-2.0	1	1	1	1	0	0
	Sum of Errors 2.0-3.0	0	0	1	1	2	2
	Sum of Errors > 3.0	0	0	0	0	0	0

Green highlighted numbers are for the better results which were obtained with more conformers. Red highlighted results are for when results should have been better with more conformers but were not. We recall that results in Figure 5.4 of Section 5.1 showed this trend for benchmark tests #7 and #8. The only difference between tests #7 and #8 in Figure 5.4 was the total number of rotamer states. Results in Table 5.6 did not show the aforementioned trend for two *different* tests but rather for the *same* test. We also would like to point out that after 30 different runs, the vanilla algorithm produced the best R.M.S.D. value for 4PTI and 4LZT but not for 1XNB. On average, the hybrid performed better and the vanilla algorithm's near-optimal solution is worse than the hybrid's.

We now show results of the second test, test #8 in Table 5.7, with an increased local residue filtering range from 3Kcal/mol to 5Kcal/mol, therefore including more rotamers. Results in green show the method for which the statistical results are superior. Similarly, results in red show the method for which statistical results are worse.

Table 5.7: GA Test #8 Hybrid and Vanilla genetic algorithm results of 3 proteins

	GA Test #8: 2eps*15Kcal/mol *Bucket Sampling* **0.0** *Occupancy Filtering*, **5Kcal/mol** *Local Residue Filtering*					
Statistics	**4PTI Hybrid**	**4PTI Vanilla**	**4LZT Hybrid**	**4LZT Vanilla**	**1XNB Hybrid**	**1XNB Vanilla**
Number of Residues	52	52	117	117	182	182
Number of Runs	30	30	30	30	30	30
Average *near-optimal*	2017.16	2018.47	2899.15	2899.13	-641.2	-640.59
The Best *near-optimal*	2017.04	2017.04	2895.92	2897.10	-647.38	-645.38
The Worst *near-optimal*	2017.78	2020.1	2907.99	2901.49	-635.89	-634.82
***near-optimal* Std. Dev.**	0.22281	0.98173	2.49641	1.25113	2.6419	2.48761
Average R.M.S.D.	0.7714	0.7429	0.8403	0.8452	0.9061	0.9067
Best R.M.S.D.	0.6963	0.7032	0.6415	0.7149	0.6252	0.4442
Worst R.M.S.D.	0.8463	0.7953	1.0132	0.9448	1.2233	1.3539
Published MCCE R.M.S.D.	0.56088		0.72376		0.85586	
MCCE 12rot./bond	0.69213		0.79606		1.23404	
MCCE 6rot./bond	0.65603		0.64532		0.84292	
Sum of Errors 1-1.5	71	75	61	76	40	40
Sum of Errors 1.5-2.0	25	15	42	44	10	16
Sum of Errors 2.0-3.0	0	0	9	8	11	10
Sum of Errors > 3.0	0	0	0	0	0	0
Average Number Conformers	2705.23	2929.2	4150.03	4270.76	2881.6	2953
Average Rotamer Generation Time	1768 s	1768 s	12543 s	12543 s	15040 s	15040 s
Average Step 3 Run Time	8628.3 s	9951.8 s	17892 s	18935 s	10926.6 s	11372.83 s
Average Step 4 Run Time	215.83 s	235.46 s	1025 s	1126.3 s	2263.6 s	2465 s
Average Total Run Time	**10815.52 s**	**11995.83 s**	**31853.49 s**	**32885.31 s**	**29012.93 s**	**29479.61 s**
1-Tail Homoscedastic T-Test of 30 Best Fitness Values	**8.5393E-10**		**0.484811551**		**0.180416733**	
1-Tail Homoscedastic T-Test of 30 R.M.S.D Values	**5.6429E-05**		**0.392858553**		**0.495314755**	

Unlike results of Table 5.5, both t-tests for the protein 4PTI on the fitness and R.M.S.D. values show statistical evidence that the hybrid method is superior to the vanilla. As for 4LZT and 1XNB, both t-tests are inconclusive. The increase in range from 3Kcal/mol to 5Kcal/mol has also considerably improved results for 4PTI.

To enhance the difference in results between 3Kcal/mol and 5Kcal/mol post-sampling filtering for local residues, we produced another Table 5.8 only for the R.M.S.D. statistics over 30 runs. Results in green show the method for which statistical results were superior. Similarly, results in red show the method for which statistical results were worse.

Table 5.8: 3Kcal/mol VS 5Kcal/mol sampling for 30 runs.

	3Kcal 4PTI HYB.	5Kcal 4PTI HYB.	3Kcal 4PTI VAN.	5Kcal 4PTI VAN.	3Kcal 4LZT HYB.	5Kcal 4LZT HYB.	3Kcal 4LZT VAN.	5Kcal 4LZT VAN.	3Kcal 1XNB HYB.	5Kcal 1XNB HYB.	3Kcal 1XNB VAN.	5Kcal 1XNB VAN.
AVE. RMSD	0.848	0.771	0.850	0.742	0.808	0.84	0.761	0.845	0.978	0.906	1.015	0.91
BEST RMSD	0.792	0.696	0.778	0.703	0.652	0.641	0.625	0.715	0.634	0.625	0.718	0.444
WORST RSMD	0.931	0.846	0.906	0.795	0.945	1.013	0.865	0.945	1.331	1.223	1.36	1.354
SUM ERRORS 1-1.5	96	71	90	75	64	61	50	76	46	40	44	40
SUM ERRORS 1.5-2.0	30	25	30	15	17	42	22	44	18	10	19	16
SUM ERRORS 2.0-3.0	0	0	0	0	9	9	5	8	13	11	15	10
SUM ERRORS > 3.0	0	0	0	0	0	0	0	0	0	0	0	0

The proteins 4PTI and 1XNB benefitted the most from the 5Kcal/mol sampling. The 4LZT protein offered its best results from the 3Kcal/mol sampling. There are also strong differences in the sums of errors between the 3Kcal/mol and the 5Kcal/mol sampling. The largest differences in sums of errors were highlighted in black. These results reinforce our assumptions that the optimal sampling range for different proteins can vary.

Both the hybrid and vanilla algorithms' near-optimal solution samples are *independent* of the bucket sampling. However, the Monte-Carlo in step 4 of MCCE is dependent upon the evolutionary sampling. To try to obtain better t-test results, we combined the 30 random runs from the two tests #7 and #8 as shown in Table 5.9 and omitted all other statistics. Results in green show the method for which statistical results were superior. Similarly, results in red show the method for which statistical results were worse.

Table 5.9: Combined GA samples of tests #7 and #8 for fitness t-test.

Statistics	4PTI Hybrid	4PTI Vanilla	4LZT Hybrid	4LZT Vanilla	1XNB Hybrid	1XNB Vanilla
Number of Residues	52	52	117	117	182	182
Number of Runs	60	60	60	60	60	60
Average Best Fitness	2017.41	2018.27	2898.81	2899.12	-640.89	-640.5
The Best Fitness	2017.04	2017.04	2895.92	2896.75	-647.38	-645.95
The Worst Fitness	2024.74	2020.36	2907.99	2902.81	-635.89	-634.82
Fitness Std. Dev.	1.38799	0.96393	2.16159	1.24921	2.55224	2.2818
1-Tail Homoscedastic T-Test of 60 Best Fitness Values	7.05853E-05		0.168986349		0.185714969	
1-Tail Homoscedastic T-Test of 30 Best Fitness Values (GA Test #7)	0.1523		0.0546		0.3789	
1-Tail Homoscedastic T-Test of 30 Best Fitness Values (GA Test #8)	8.5393E-10		0.484811551		0.180416733	

The difference of each protein's fitness t-tests between test #7 and test #8 are considerable. The results of Table 5.9 suggest the need for more random runs. For the time being, we can statistically argue on the superiority of the hybrid algorithm for at least 1 out of 3 proteins.

5.3 Hybrid Algorithm Convergence and Diversity Properties

We would like to address the convergence property of our stochastic search algorithm. It is important to demonstrate its capability to maintain a diverse population of chromosomes while convergence occurs and offer some consistency in achieving near-optimality. As previously stated in Chapter 4, we designed our genetic operators to avoid getting stuck in local minima as much as possible. We constantly introduced new genetic material to ensure continuous genetic diversity. For a more detailed visual proof of genetic diversity, all of the experiments' histograms of the last genetic population prior to dynamic bi-directional evolutionary sampling are available in Appendix A. Figure 5.6 confirms this diversity property for the egg white lysozyme protein (4LZT).

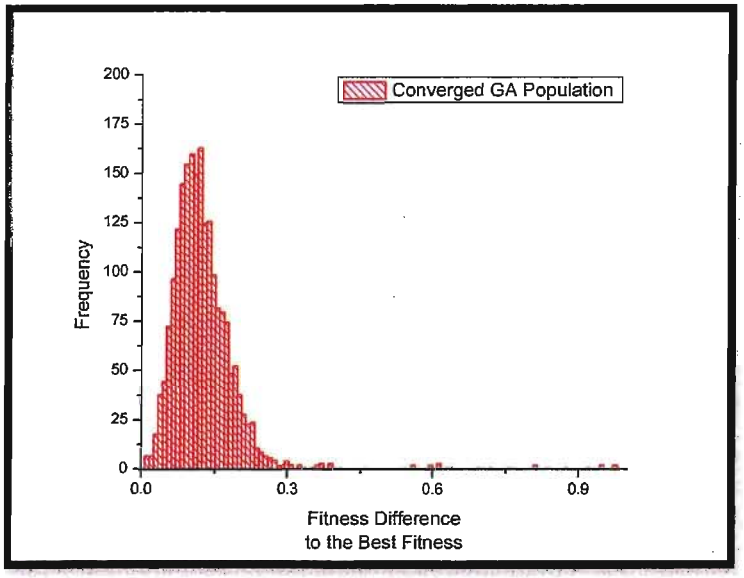

Figure 5.6: Histogram of converged population.
Showing 2,072/3,000 chromosomes.

Not all of the proteins' optimizations were successful in converging to the specified dE value of 0.15, as shown below in Table 5.10. This phenomenon could be attributed to the nature of the studied protein, the percentage of population looked at for convergence (70% of the population for dE calculation) and the number of generations allowed.

Table 5.10: Convergence of protein data set prior to evolutionary sampling

PDB	Converged? (dE<0.15)	dE	Number Generations /450	Initial Best Fitness	Last Best Fitness
1PGB	NO	0.2171723	450	-67.2091675	-95.4542313
1LE2	NO	12.1464052	450	-301.5581665	-301.5581665
1NFN	YES	0.1444521	125	-27.8614082	-275.9095154
1GS9	YES	0.1253164	145	-35.0638504	-315.7090759
1IGD	NO	7.2615666	450	-122.9152603	-122.9152603
4PTI	NO	0.3811651	450	-49.5985222	-127.3982315
1IG5	NO	11.6668739	450	-160.6961365	-160.6961365
4LZT	YES	0.1497687	123	-73.5593414	-294.6210632
1DWR	NO	103.3649368	450	-473.3876038	-473.3876038
1A6K	NO	40.0120506	450	-422.8645325	-422.8645325
3RN3	YES	0.1138497	119	-58.7456169	-276.2914429

1GOA	YES	0.1355239	163	-244.5017548	-309.4340820
1RGG	YES	0.1410044	67	-29.5756035	-150.0839691
1I0V	YES	0.1440886	79	-46.7632523	-158.3863525
1DG9	YES	0.1439048	190	-259.0524902	-344.9213562
1H4G	YES	0.1184607	207	-77.7751312	-516.4929810
1XNB	**NO**	**136.9107819**	**450**	**-561.0968628**	**-561.0968628**
1KXI	NO	0.4069000	450	-18.8874893	-112.1806870
2CI2	NO	0.3619624	450	15.2777271	-101.5453949
2CPL	**NO**	**132.2505951**	**450**	**-475.7062988**	**-475.7062988**
1HNG	YES	0.1475331	314	42.8444138	-366.1554565
3EBX	NO	0.4077300	450	-22.7107182	-132.0970459
1BEO	NO	0.2613926	450	-10.6234322	-162.4570312
1PPF	NO	0.5193815	450	-3.9501691	-69.8858414

Out of 24 proteins, 10 of them were able to converge below a dE of 0.15. Out of the 14 that did not converge, 7 of those were below a dE of 1.0. Setting our dE convergence criterion to 1.0 would have yielded 17 out of 24 converged proteins. If we pay close attention to the remaining non-converged proteins 1LE2, 1IGD, 1IG5, 1DWR, 1A6K, 1XNB, and 2CPL in bold in Table 5.10, we observed that after 450 generations of evolution, no better conformation was found. In other words, the best conformation was the original structure. These results reinforce the fact that side-chain packing is protein specific.

5.4 Conclusion and Future Work

We are satisfied with our results and algorithm performances. We have shown that our adapted hybrid genetic algorithm / *pseudo* simulated annealing performed well and offered speed up of magnitude as opposed to the current method of MCCE. We also showed that side-chain packing is not only a problem of approximating the optimal solution of the protein. It is also a problem of sampling for other conformations that can coexist in the molecular ensemble of the protein at a given temperature. This sampling problem requires further analysis for the prediction of proteins' pKa's. Sampling in this high-dimensional search-space is difficult and since side-chain packing was shown to be

NP-Hard, we feel that we have contributed to research in this field. Notwithstanding the latter, we have also proposed a novel technique of dynamic bi-directional evolutionary sampling as a statistical sampler for rotameric states and can be applied to different problems. Due to MCCE's inability to consistently sample pKa's in step 4, we find it difficult to properly establish the quality of our algorithm *per se*. For all we know, based on the observed results' trends, our method could be giving MCCE the "perfect" set of rotameric states but not correctly predicting proteins' pKa's. The evaluation function in our algorithm did not include electrostatic forces. MCCE calculates electrostatic interactions as in step 3. This calculation performs many useless computations and requires a very large amount of RAM. Ideally, we would like to offload these electrostatic calculations on a GPU for very fast and optimized computations, and add this extra energetic term as part of our fitness evaluation function. Doing so would force us to include ionization states for all amino-acid residues and solve the Poisson-Boltzmann equation (electrostatic) for each chromosome of the population. We have already begun a preliminary implementation of this algorithm and we are currently looking at different ways of accomplishing this feat. We would like to emphasize that our evolutionary sampling technique also calls for mathematical proofs for its correctness as a statistical sampler and will necessitate several months of work on its own. As Marc N Offman et. al. suggested [47], the idea of alternating selective pressure, however dissimilar to our novel evolutionary sampler it might be, requires further research so that in the end, consistent prediction of solutions can be obtained. Parallelization of our methodology, or any method as a matter of fact, for side-chain packing of proteins with thousands of residues - Photosystem II for example - for pKa prediction remains an open problem as suggested by T. Akutsu. et. al. [51].

Bibliography

1. *NP-hardness results for protein side-chain packing.* **Akutsu, T.** 1997, Genome Inform., 8, pp. 180-186.

2. *Complexity of protein folding.* **Fraenkel, Aviezri S.** 6, New York : Springer, November 1993, Bulletin of Mathematical Biology, Vol. 55. 0092-8240 (Print) 1522-9602 (Online).

3. *Correction to "A Formal Basis for the Heuristic Determination of Minimum Cost Paths".* **Hart, Peter E., Nilsson, Nils J. and Raphael, Bertram.** 37, New York : ACM, 1972, Vol. 0. 0163-5719.

4. *A note on two problems in connexion with graphs.* **Dijkstra, E. W.** 1, Berlin : Springer/Heidelberg, 1959, Vol. 1. 0029-599X.

5. *No Free Lunch Theorems for Optimization.* **Wolpert, D. H. and Macready, W. G.** 1997, IEEE Transactions on Evolutionary Computation, 1, p. 67.

6. *On the futility of blind search: An algorithmic view of "no free lunch".* **Culberson, Joseph C.** 2, Edmonton : MIT Press, 1998, Evolutionary Computation, Vol. 6, pp. 109-127. ISSN:1063-6560.

7. *A Representation for the Adaptive Generation of Simple Sequential Programs.* **Cramer, Nichael Lynn.** [ed.] John J. Grefenstette. Pittsburgh : s.n., 1985. PROCEEDINGS OF AN INTERNATIONAL CONFERENCE ON GENETIC ALGORITHMS AND THEIR APPLICATIONS. pp. 183-187. http://homepages.sover.net/~nichael/nlc-publications/icga85/index.html.

8. **Koza, John R.** *Genetic Programming: A Paradigm for Genetically Breeding Populations of Computer Programs to Solve Problems.* Computer Science Department, Stanford University. Palo Alto : s.n., 1990. STAN-CS-90-1314.

9. **Banzhaf, Wolfgang, et al.** *Genetic Programming: An Introduction: On the Automatic Evolution of Computer Programs and Its Applications.* USA : Morgan Kaufmann, 1998. 1-55860-510-X.

10. *Design & Implementation of Parallel Linear Genetic Programming on the IBM Cell Processor.* **Comte, Pascal.** Montreal : ACM Online Library, July 2009. GECCO'09 CIGPU.

11. *Optimization by Simulated Annealing.* **Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P.** 1983, Science. New Series, 220 (4598), pp. 671-680.

12. *A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm.* **Cerny, V.** 1985, Journal of Optimization Theory and Applications, 45, pp. 41-51.

BILIOGRAPHY

13. *Tabu Search — Part I.* **Glover, F.** 3, 1989, ORSA Journal on Computing, Vol. 1. p. 190-206.

14. *Tabu Search — Part II.* **Glover, F.** 1, 1990, ORSA Journal on Computing, Vol. 2. p. 4-32.

15. *A new optimizer using particle swarm theory.* **Kennedy, J. and Eberhart, R. C.** Nagoya, Japan : s.n., 1995. Proceedings of the Sixth International Symposium on Micromachine and Human Science. pp. p. 39-43.

16. *Particle swarm optimization.* **Kennedy, J. and Eberhart, R. C.** Piscataway, New Jersey : s.n., 1995. Proceedings of IEEE International Conference on Neural Networks. pp. 1942-1948.

17. *The Monte Carlo Method.* **Metropolis, N. and Ulam, S.** Journal of the American Statistical Association , Vol. 44, pp. 335-341. DOI: 10.2307/2280232.

18. *Hybrid genetic algorithm and simulated annealing approach for the optimization of process plans for prismatic parts.* **Li, WD, Ong, SK and Nee, YC.** 2002, International Journal of Production Research, Vol. 40, pp. 1899-1922.

19. *Solving a timetabling problem using hybrid genetic algorithms.* **Kragelund, LV.** 1997, Software—Practice and Experience, Vol. 27, pp. 1121-1134.

20. *The Algorithm For Berth Scheduling Problem by The Hybrid Optimization Strategy GASA.* **Han, Mei, Li, Ping and Sun, Junging.** 2006. ICARCV. Vol. 5, pp. 1-4.

21. *A Genetic Algorithm Hybrid for Constructing Optimal Response Surface Designs.* **Drain, David, Carlyle, W. Matthew and al., et.** 2003, Quality and Reliability Engineering International, Vol. 20, No. 7, pp. 637-650.

22. *MCCE2: Improving protein pKa calculations with extensive side-chain rotamer sampling.* **Song, Y., Gunner, M. R. and Mao, J.** 2009, Journal Comput. Chem., 30(14), pp. 2231-2247.

23. *Simulation of genetic systems by automatic digital computers I. Introduction.* **Fraser, Alex.** 1957, Aust. J. Biol. Sci., 10, pp. 484-491.

24. **Fraser, Alex and Donald, Burnell.** *Computer Models in Genetics.* New York : McGraw-Hill, 1970.

25. **Holland, John H.** *Adaptation in Natural and Artificial Systems.* Ann Arbor : University of Michigan Press, 1975. 978-0262581110.

26. *New Encoding/Converting Methods of Binary GA/Real-Coded GA.* **Kim, Jong-Wook and Kim, Sang Woo.** 6, IEICE Trans. Fundamentals, Vols. E88-A, pp. 1554-1564.

27. *Exact Algorithms for NP-Hard Problems: A Survey.* **Woeginger, G. J.** s.l. : Springer, Combinatorial Optimization – Eureka, You Shrink! Lecture notes in computer science, Vol. 2570, pp. 185-207.

BILIOGRAPHY

28. **Nichols, R.** Quenching and tempering of welded carbon steel tubulars; Minimal process variations to obtain a uniform product. *thefabricator.com.* [Online] Fabricators & Manufacturers Association, Intl., July 29, 2001. [Cited: 03 04, 2010.] http://www.thefabricator.com/article/tubepipefabrication/quenching-and-tempering-of-welded-carbon-steel-tubulars.

29. *Optimization by Simulated Annealing.* **Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P.** 1983, Science. New Series, 220 (4598), pp. 671-680.

30. *Simulated annealing: A proof of convergence.* **Granville, V. and et., al.** 1994, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(6), pp. 652-656.

31. *A pragmatic approach to structure based calculation of coupled proton and electron transfer in proteins.* **Gunner, M. R. and Alexov, E.** 2000, Biochem and Biophys. Acta., 1458, pp. 63-87.

32. *Incorporating protein conformational flexibility into pH-titration calculations: Results on T4 Lysozyme.* **Gunner, M. R. and Alexov, E.** 1997, Biophysics Journal, 74, pp. 2075-2093.

33. *Combining conformational flexibility and continuum electrostatics for calculating pKa's in proteins.* **Gunner, M. R., Alexov, E. and Georgescu, R. E.** 2002, Biophysics Journal, 83, pp. 1731-1748.

34. *pH dependence of heme electrochemistry in cytochromes investigated by multiconformation continuum electrostatics calculations.* **Hauser, K., Mao, J. and Gunner, M. R.** 2004, Biopolymers, 42, pp. 51-54.

35. *How cytochromes with different folds control heme redox potentials.* **Mao, J., Hauser, K. and Gunner, M. R.** 2003, Biochemistry, 42, pp. 9829-9840.

36. *Energetics of Quinone-Dependent Electron and Proton Transfers in Rhodobacter sphaeroides Photsynthetic Reaction Centers.* **Zhu, Z. and Gunner, M. R.** 2005, Biochemistry, 44, pp. 82-96.

37. *Calculation of proton transfers in Bacteriorhodopsin bR and M intermediates.* **Song, Y., Mao, J. and Gunner, M. R.** 2003, Biochemistry, 42, pp. 9875-9888.

38. *Do Intelligent Configuration Search Techniques Outperform Random Search for Large Molecules?* **Judson, R. S., et al.** No. 2, New York : John Wiley & Sons, 1992, International Journal of Quantum Chemistry, Vol. 44, pp. 277-290. ISSN: 0020-7608.

39. *Conformational Searching Methods for Small Molecules. II. Genetic Algorithm Approach.* **Judson, R. S., et al.** 1993, Journal of Computational Chemistry, Vol. 14, No.11, pp. 1407-1414.

40. **Grenfenstette, J J and Schraudoplh, N N.** GENESIS 1.2ucsd. [Online] ftp://ftp.aic.nrl.navy.mil/pub/galist/source-code/ga-source/gaucsd 12.tar.

41. **Associates, Tripos.** SYBYL v.5.4.1. St. Louis, MO : s.n., 1991.

42. **Dammkoehler, R. A., et al.** 1989, Journal Comp.-Aided Mol. Design, 3,3.

43. **Allen, F. H., et al.** 1979, Acta Crystallogr., B, 35, 2331.

44. *Genetic Algorithms in Conformational Analysis.* **Nair, Nikhil and Goodman, Jonathan M.** 2, s.l. : American Chemical Society, 1998, Journal of Chemical Information and Computer Sciences, Vol. 38, pp. 317-320. ISSN: 0095-2338.

45. *A Genetic Algorithm for Conformational Search of Organic Molecules: Implications for Materials Chemistry.* **Keser, Milan and Stupp, Samuel I.** 1998, Computers Chem. Vol. 22, No. 4, pp. 345-351.

46. *Using genetic algorithm to design protein sequence.* **Scott, Luis P.B., Chahine, Jorge and Ruggiero, José R.** 1, s.l. : Elsevier, June 15, 2008, Applied Mathematics and Computation, Vol. 200, pp. 1-9.

47. *Alternating evolutionary pressure in a genetic algorithm facilitates protein model selection.* **Offman, Marc, Tournier, Alexander and Bates, Paul.** 34, 2008, BMC Structural Biology, Vol. 8. ISSN: 1472-6807.

48. *Finite population models of dynamic optimization with stochastically alternating fitness functions.* **Liekens, AML, ten Eikelder, HMM and Hilbers, PAJ.** s.l. : IEEE, 2003. IEEE Conference on Evolutionary Computation. Vol. 2, pp. 838-845.

49. *Prediction of Side-chain Conformation by Packing Optimization.* **Lee, Christopher and Subbiah, S.** 1991, Journal Molecular Biology, 217, pp. 373-388.

50. **Horvath, Csaba and al., et.** 1976, Journal Chromatogr., 125, pp. 129-156.

51. *Protein side-chain packing problem: a maximum edge-weight clique algorithmic approach.* **Akutsu, T., et al.** 2005, Journal of Bioinformatics and Computational Biology, Vol. 3, No. 1, pp. 103-126.

52. *A cooperative combinatorial swarm optimization algorithm for side-chain packing.* **Lapizco-Encinas, Grecia, Kingsford, Carl and Reggia, James.** 2009. IEEE Swarm Intelligence Symposium. pp. 22-29.

53. *A discrete binary version of the particle swarm algorithm.* **Kennedy, J and Eberhart, R C.** s.l. : IEEE, 1997. IEEE International Conference on Systems, Man, and Cybernetics. Vol. 5, pp. 4104-4108.

54. *Backbone-dependent rotamer library for proteins application to side-chain prediction.* **Dunbrack, R. L. and Karplus, M.** 1993, Journal of Molecular Biology, 230(2), pp. 543-574.

55. *Solving and analyzing side-chain positioning problems using linear and integer programming.* **Kingsford, C., Chazelle, B. and Singh, M.** 2005, Bioinformatics, 21(7), pp. 1028-1039.

BILIOGRAPHY

56. *Protein design is NP-Hard.* **Pierce, N. A. and Winfree, E.** 2002, Protein Eng., 15(10), pp. 779-782.

57. *The Penultimate Rotamer Library.* **Lovell, Simon C., et al.** 2000, PROTEINS: Structure, Function, and Genetics, 40, pp. 389-408.

58. *Do Intelligent Configuration Search Techniques Outperform Random Search for Large Molecules?* **Judson, R. S., et al.** 1992, Int. Journal Quantum Chem., 44, p. 227.

59. **Gould, Stephen Jay.** *Wonderful Life: The Burgess Shale and the Nature of History.* s.l. : W. W. Norton & Co., 1989. ISBN 0-393-02705-8.

60. *On the ergodicity properties of some adaptive MCMC algorithms.* **Andrieu, Christophe and Moulines, Eric.** 2006, The Annals of Applied Probability, Vol. 16, No. 3, pp. 1462-1505.

61. *Quantitative Non-geometric Convergence Bounds for Independence Samplers.* **Roberts, Gareth O. and Rosenthal, Jeffrey S.** 2009, Methodology and Computing in Applied Probability, p. to appear.

Appendix A

Population of the Genetic Algorithm Prior to Sampling

Convergence Criterion: dE of population ≤ 0.15

Maximum Number of Generations without convergence: 450 generations

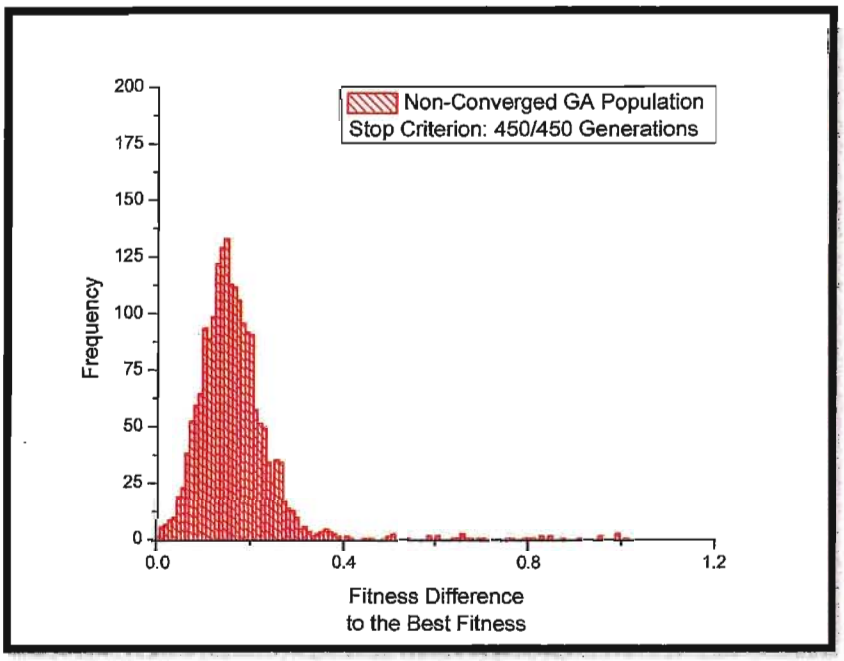

Figure A.1: Protein 1PGB for 1998/3000 chromosomes
dE = 0.2171723

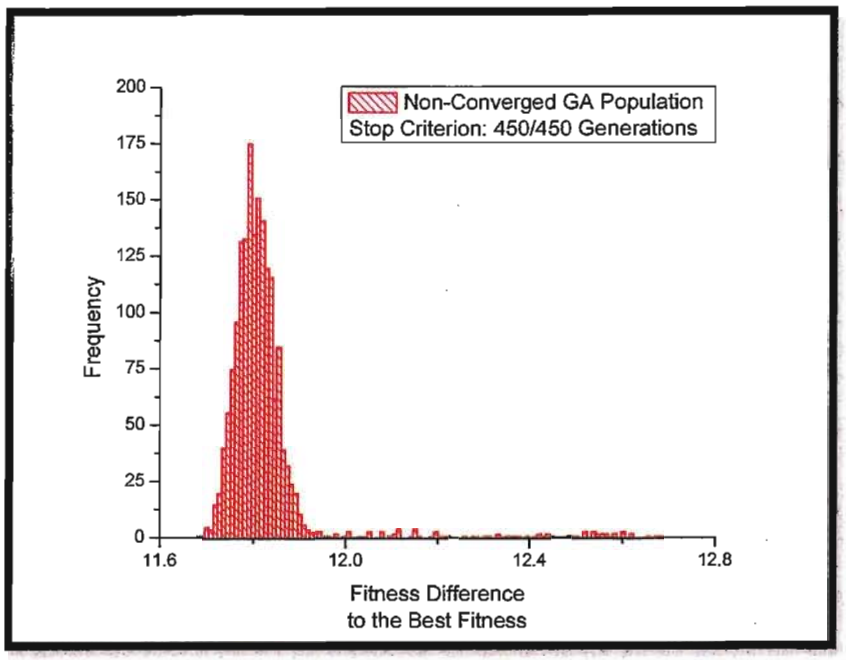

Figure A.2: Protein 1LE2 for 1790/3000 chromosomes
dE = 12.1464052

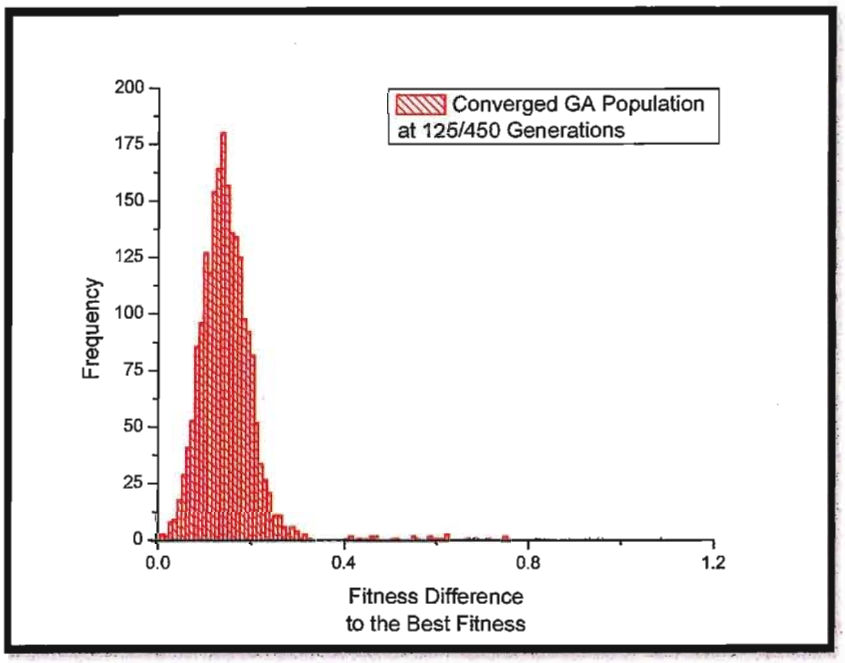

Figure A.3: Protein 1NFN for 2136/3000 chromosomes
dE = 0.1444521

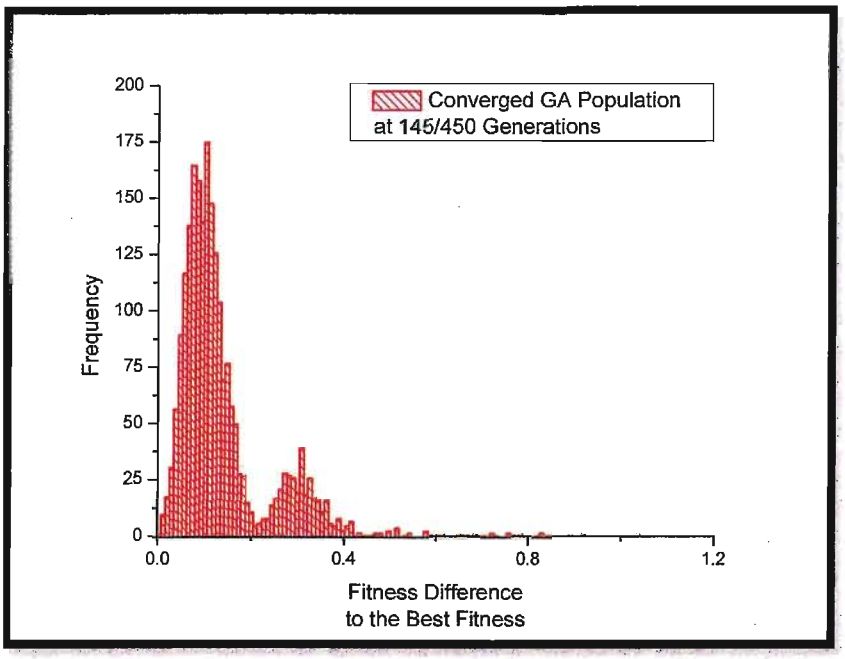

Figure A.4: Protein 1GS9 for 2219/3000 chromosomes
dE = 0.1253164

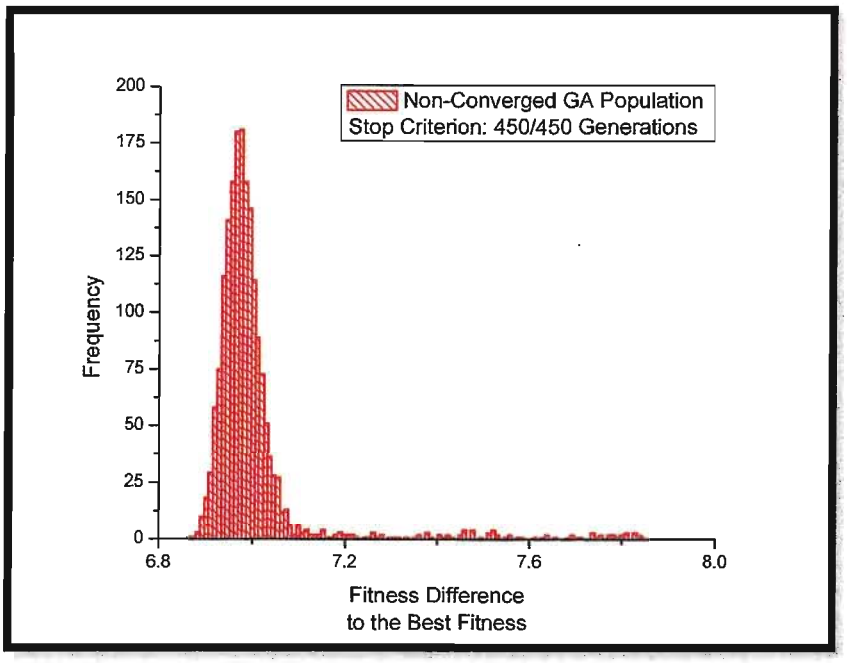

Figure A.5: Protein 1IGD for 1845/3000 chromosomes
dE = 7.2615666

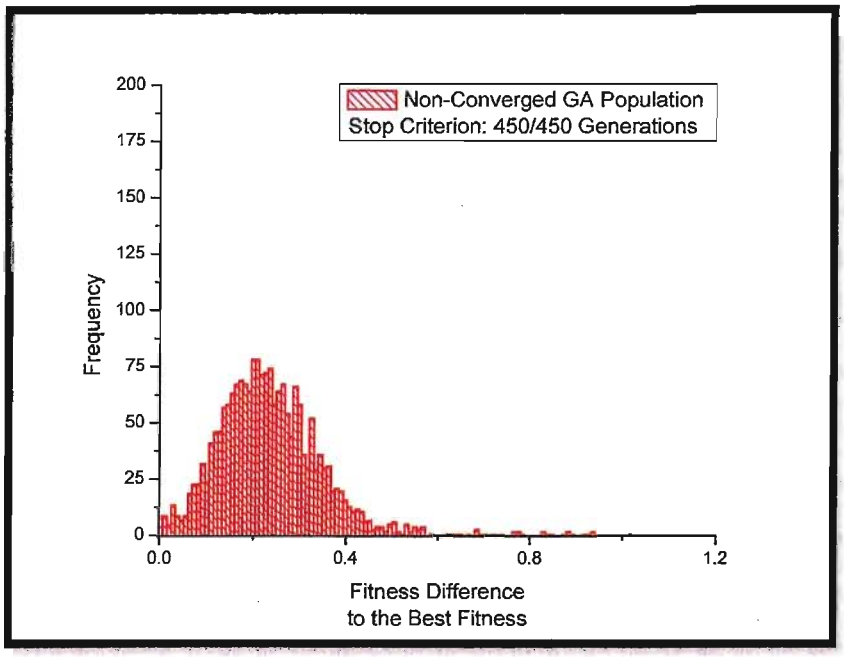

Figure A.6: Protein 4PTI for 2004/3000 chromosomes
dE = 0.3811651

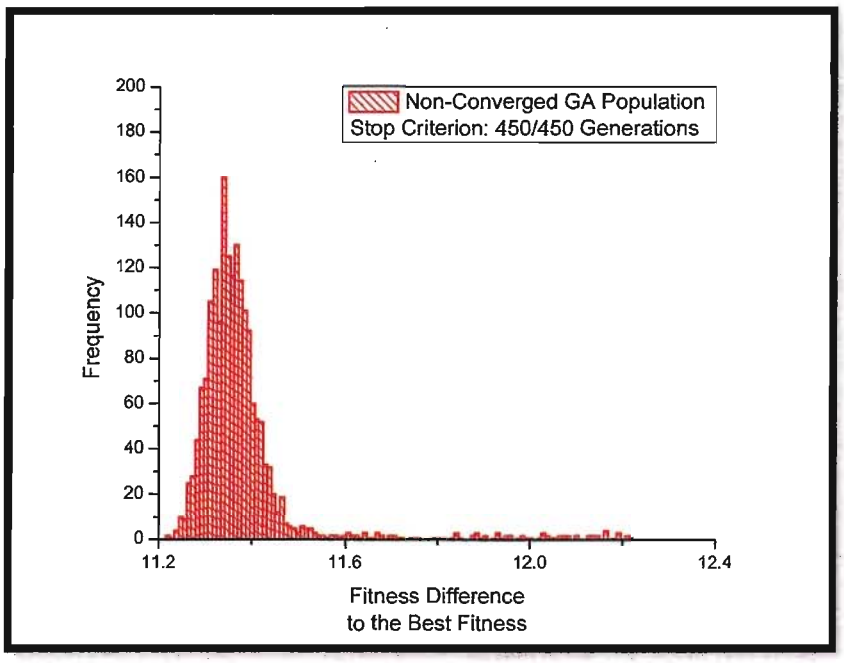

Figure A.7: Protein 1IG5 for 1843/3000 chromosomes
dE = 11.6668739

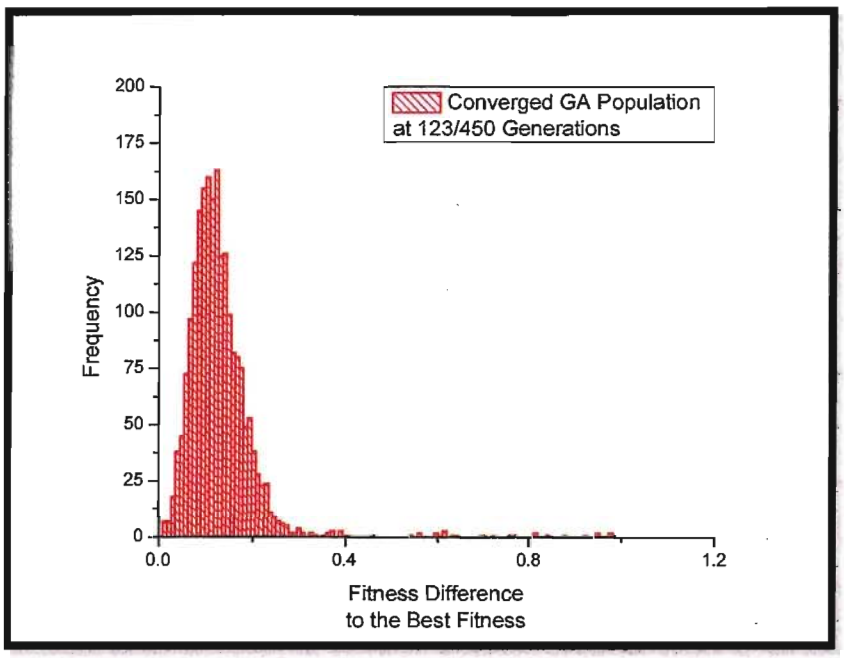

Figure A.8: Protein 4LZT for 2072/3000 chromosomes
dE = 0.1497687

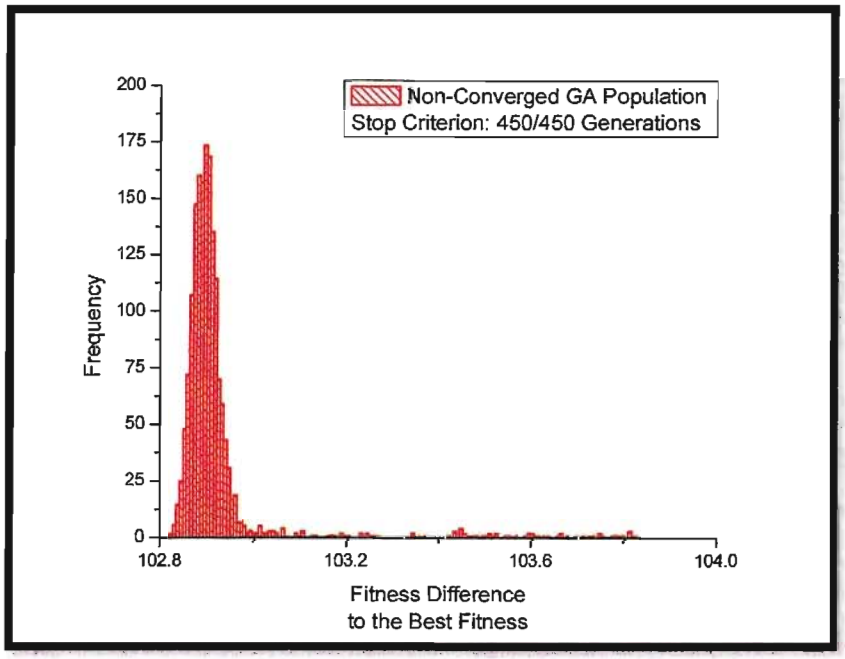

Figure A.9: Protein 1DWR for 1698/3000 chromosomes
dE = 103.3649368

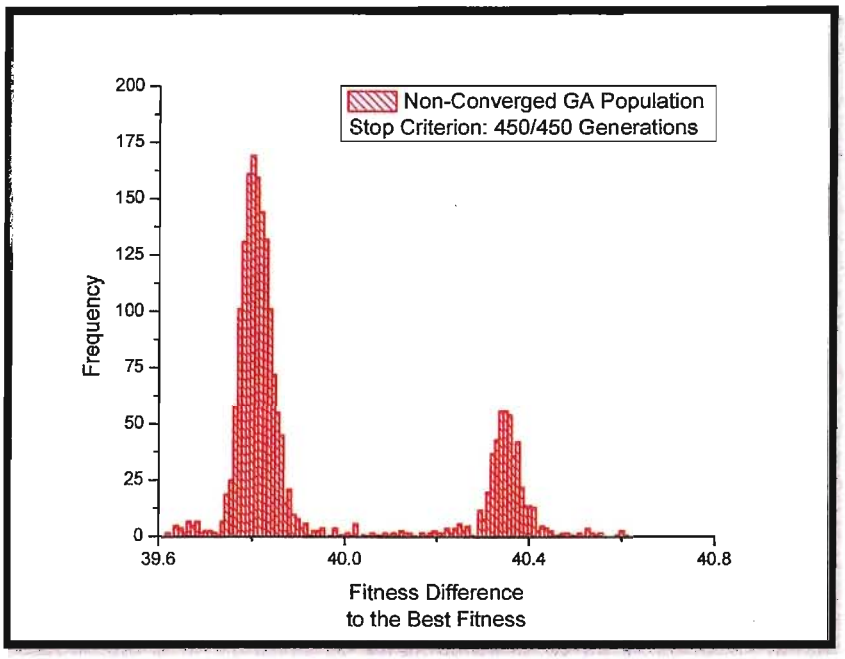

Figure A.10: Protein 1A6K for 2037/3000 chromosomes
dE = 40.0120506

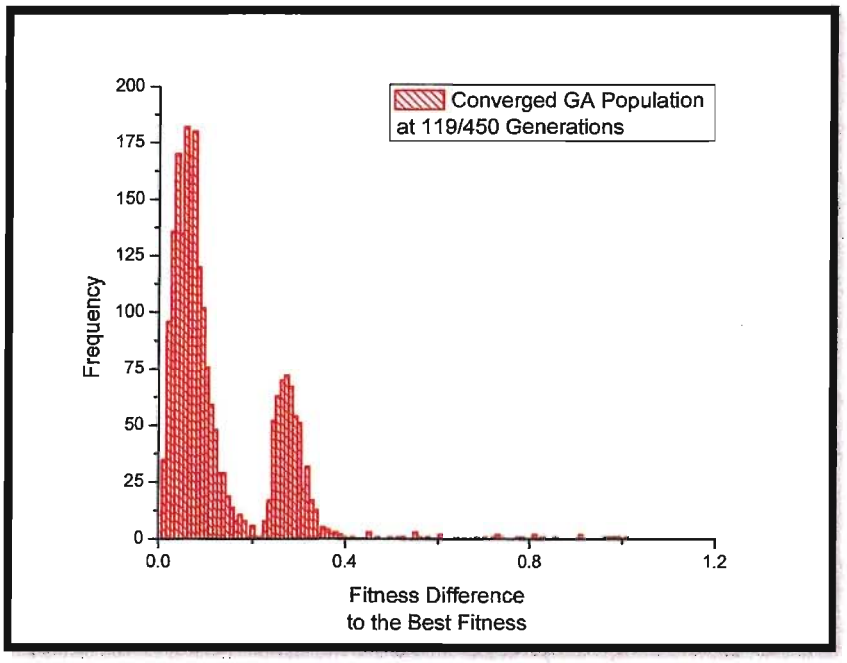

Figure A.11: Protein 3RN3 for 2242/3000 chromosomes
dE = 0.1138497

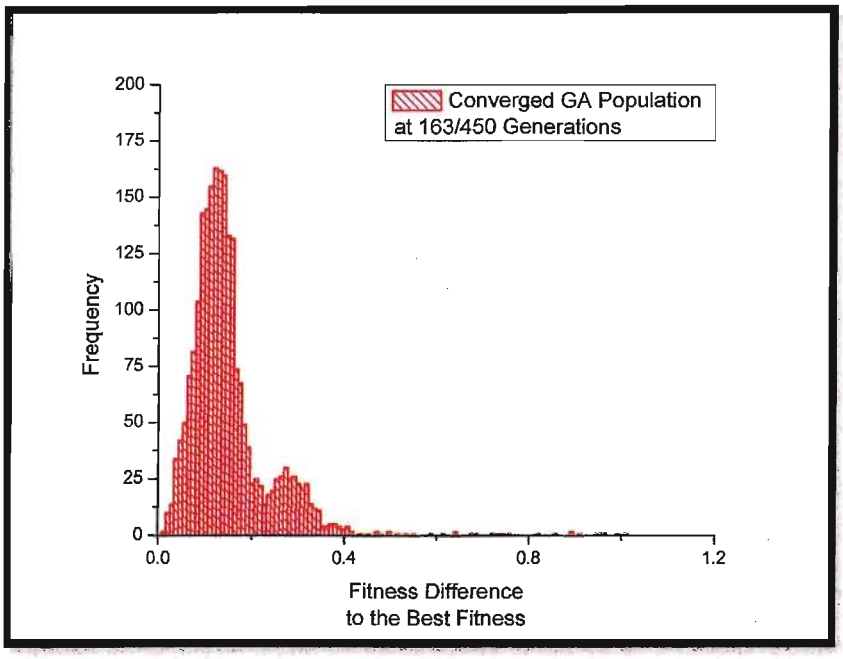

Figure A.12: Protein 1GOA for 2252/3000 chromosomes
dE = 0.1355239

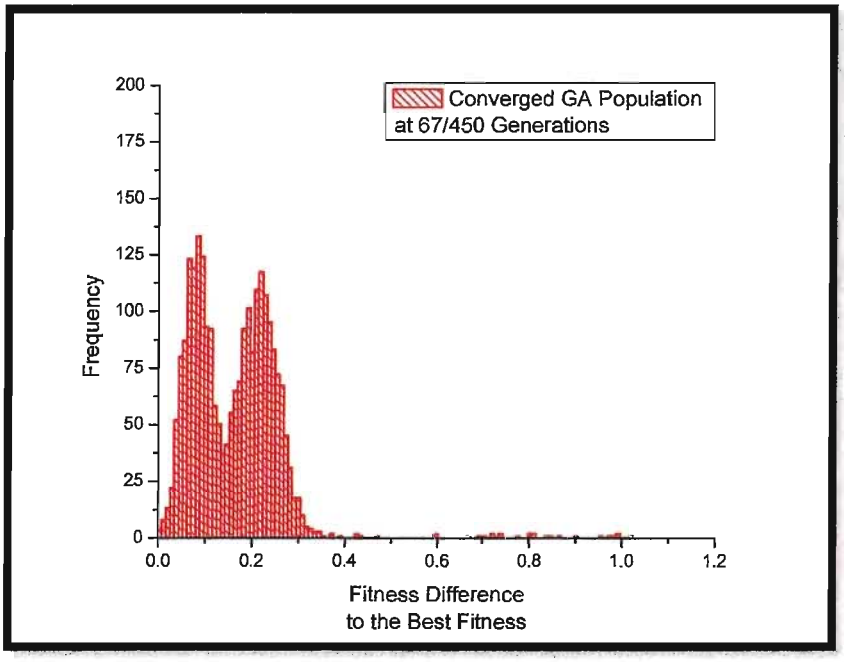

Figure A.13: Protein 1RGG for 2418/3000 chromosomes
dE = 0.1410044

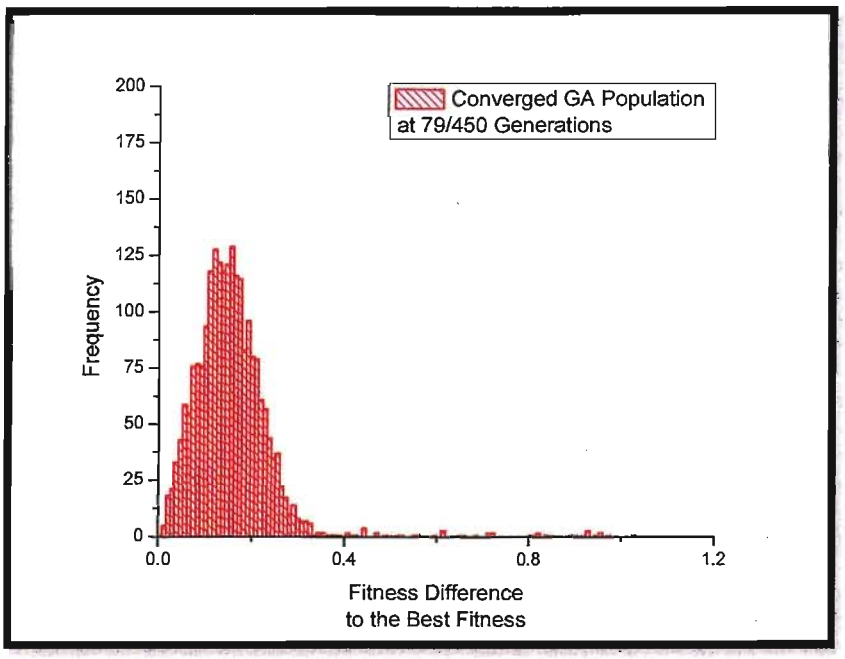

Figure A.14: Protein 1I0V for 2241/3000 chromosomes
dE = 0.1440886

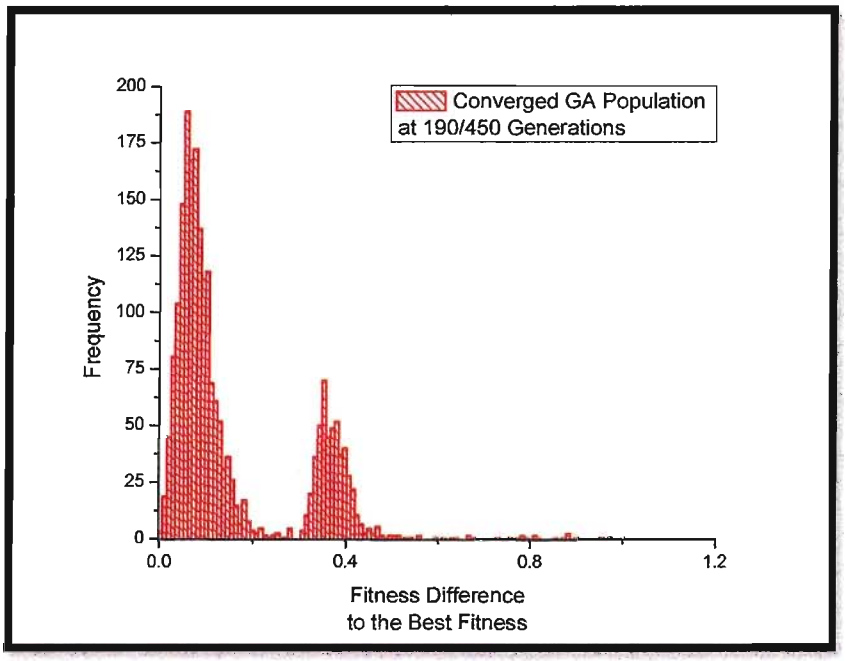

Figure A.15: Protein 1DG9 for 2186/3000 chromosomes
dE = 0.1439048

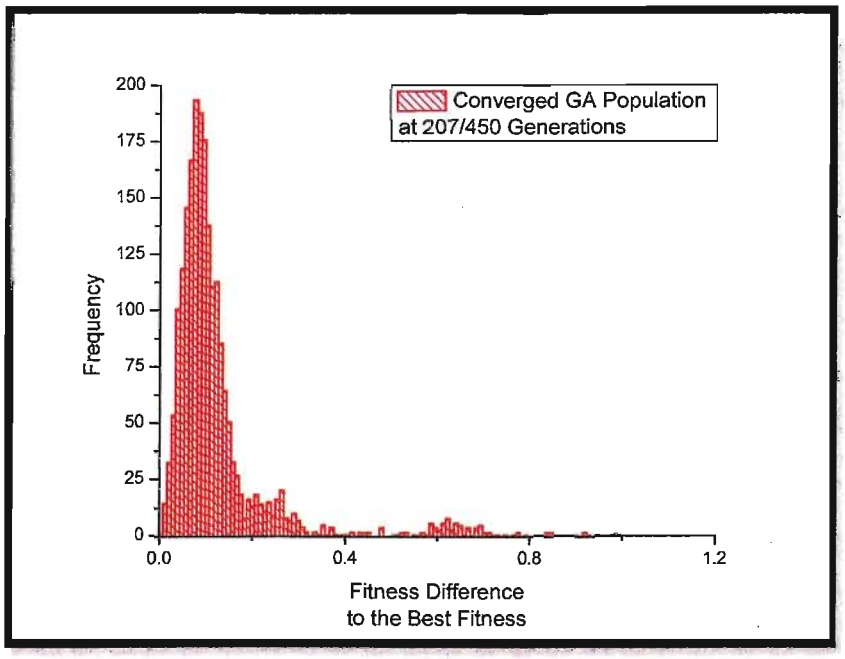

Figure A.16: Protein 1H4G for 2135/3000 chromosomes
dE = 0.1184607

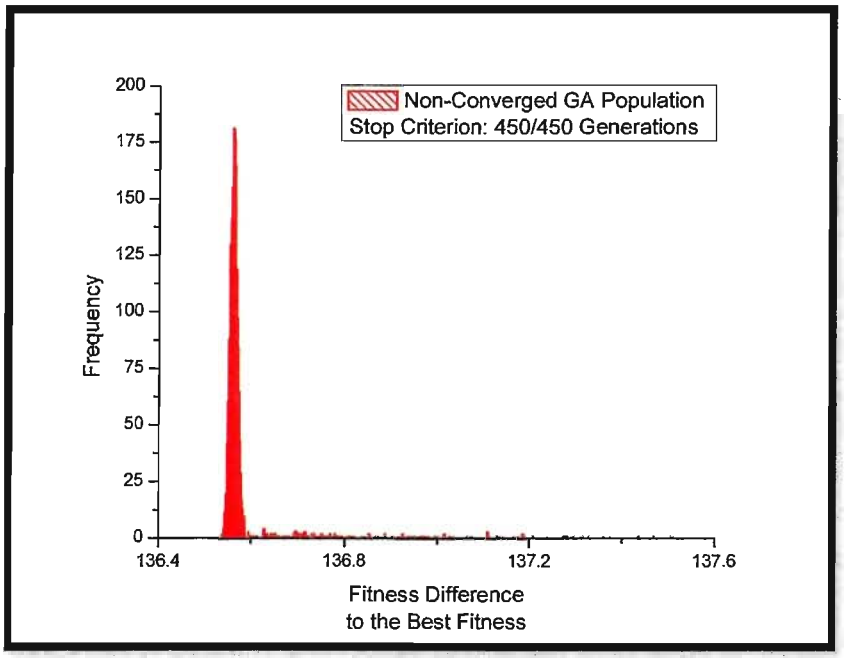

Figure A.17: Protein 1XNB for 1813/3000 chromosomes
dE = 136.9107819

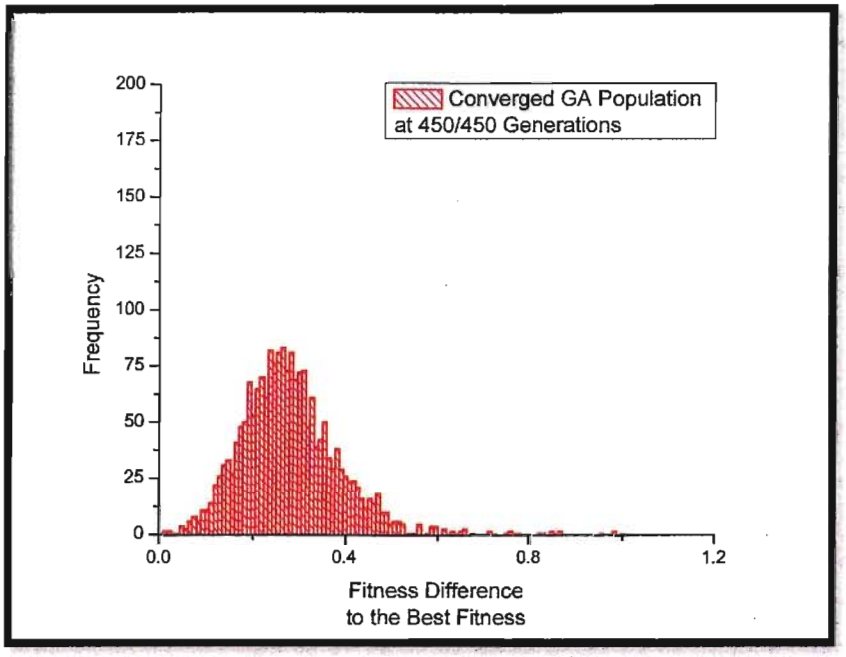

Figure A.18: Protein 1KXI for 1986/3000 chromosomes
dE = 0.4069000

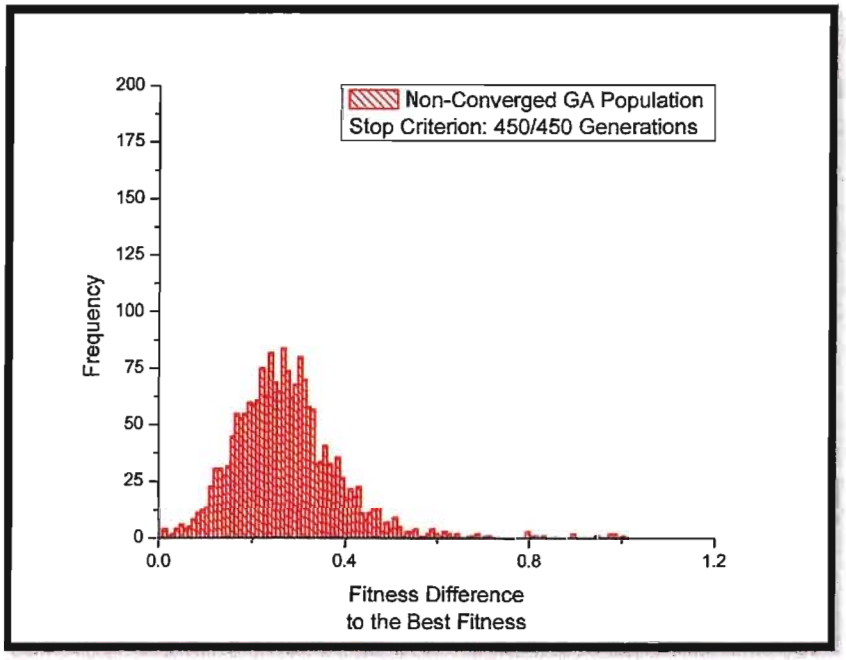

Figure A.19: Protein 2CI2 for 1960/3000 chromosomes
dE = 0.3619624

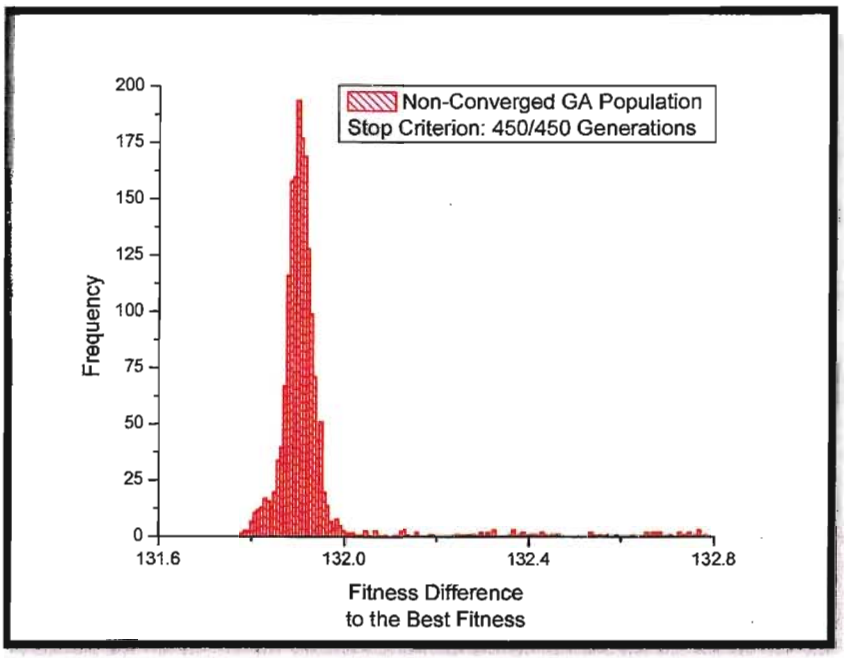

Figure A.20: Protein 2CPL for 1772/3000 chromosomes
dE = 132.2505951

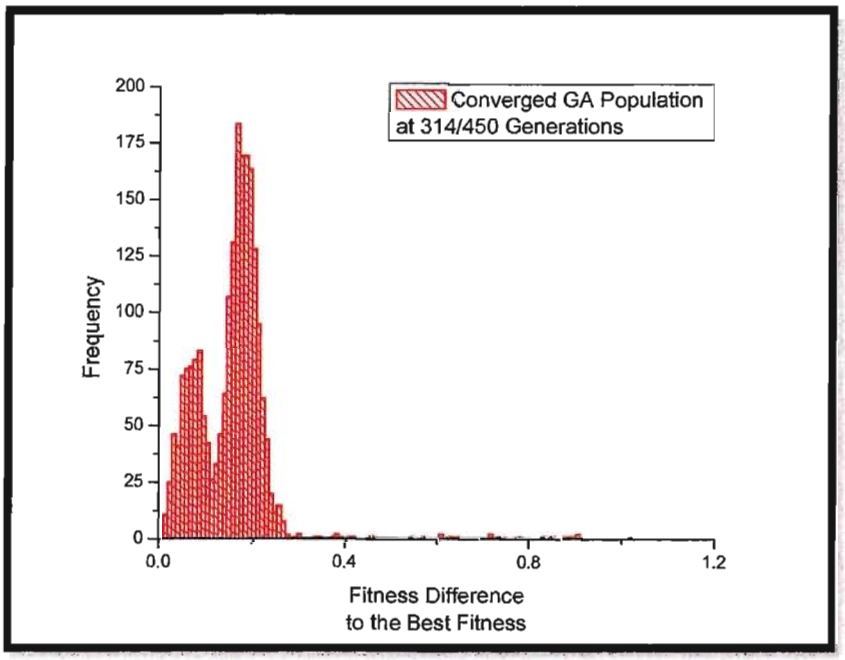

Figure A.21: Protein 1HNG for 2119/3000 chromosomes
dE = 0.1475331

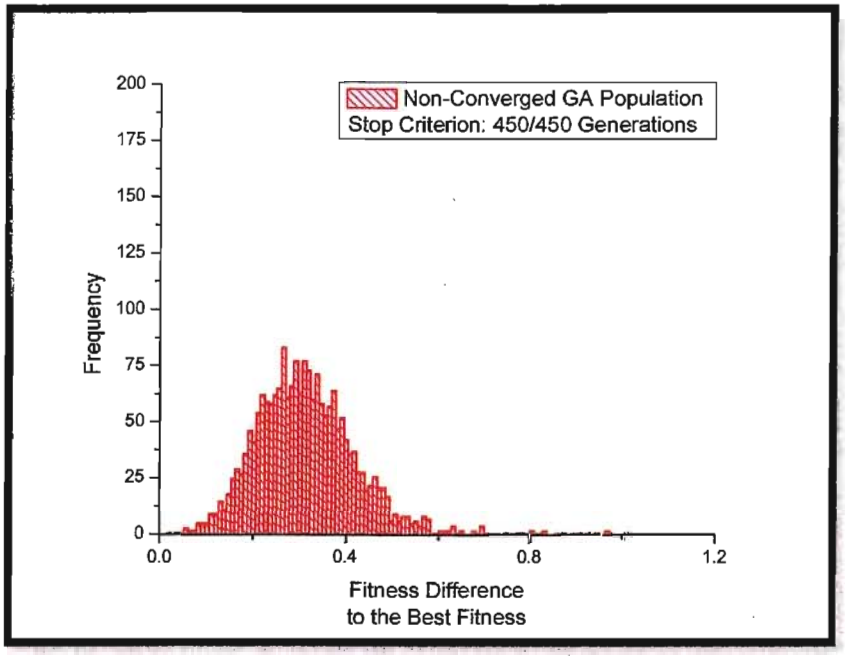

Figure A.22: Protein 3EBX for 2020/3000 chromosomes
dE = 0.4077300

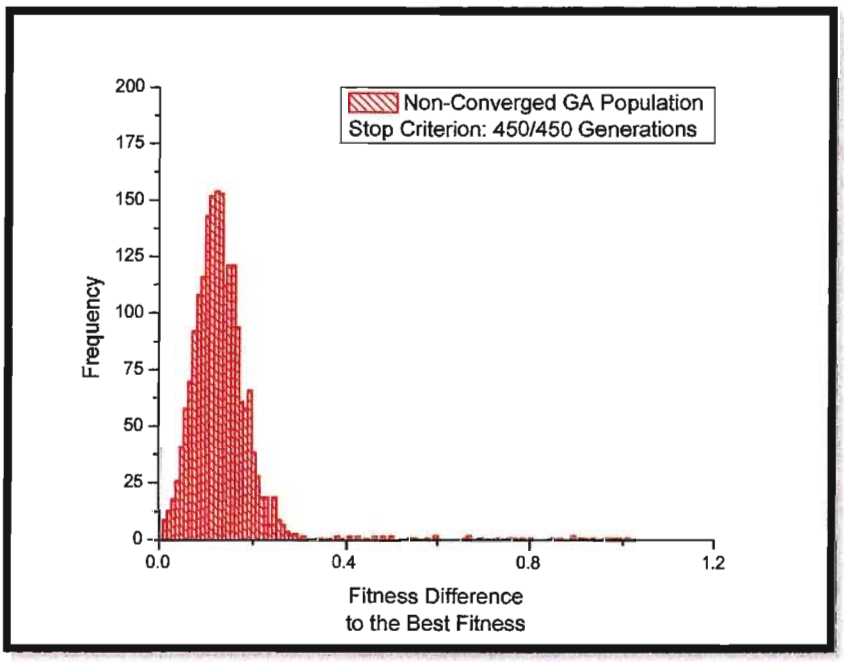

Figure A.23: Protein 1BEO for 1989/3000 chromosomes
dE = 0.2613926

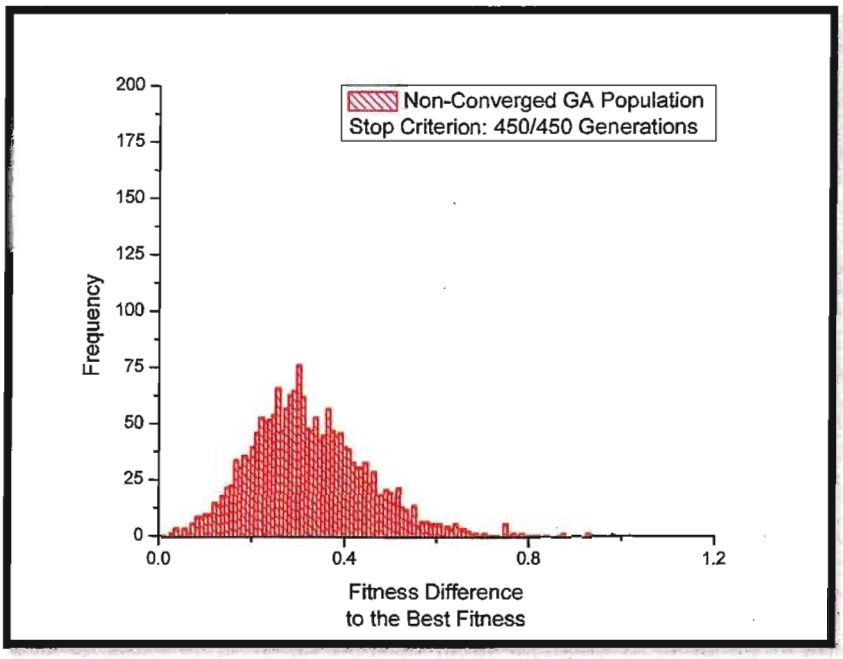

Figure A.24: Protein 1PPF for 1969/3000 chromosomes
dE = 0.5193815

Appendix B

Statistical Results of Protein Data Set

1PGB	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.566274	-0.286666667	0.50549363	1	0	0	0	2
#1 MCCE	0.581309	-0.323333333	0.500044808	2	0	0	0	2
#2 MCCE	0.755787	-0.277733333	0.727578122	2	1	0	0	2
#3 GA	0.661169	-0.2872	0.616436325	3	0	0	0	2
#4 GA	0.61509	-0.234933333	0.588407836	1	0	0	0	2
#5 GA	0.635643	-0.2642	0.598426532	2	0	0	0	2
#6 GA	0.598991	-0.2154	0.578538652	2	0	0	0	2
#7 GA	0.560239	-0.2686	0.508908327	2	0	0	0	2
#8 GA	0.447279	-0.2254	0.399892807	0	0	0	0	2

Table B.1: Protein 1PGB

1LE2	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	1.174126	-0.7	1.01816829	1	0	1	0	0
#1 MCCE	0.620228	-0.325714286	0.570109847	1	0	0	0	0
#2 MCCE	0.720187	-0.589285714	0.447190755	2	0	0	0	0
#3 GA	0.708345	-0.204114285 7	0.732631757	1	0	0	0	0
#4 GA	0.723946	-0.153714286	0.764112122	2	0	0	0	0
#5 GA	0.707411	-0.203571429	0.731769968	1	0	0	0	0
#6 GA	0.715231	-0.186	0.745957103	1	0	0	0	0
#7 GA	0.609589	-0.375714286	0.518500953	1	0	0	0	0
#8 GA	0.690164	-0.451428571	0.563881151	0	1	0	0	0

Table B.2: Protein 1LE2

APPENDIX B - STATISTICAL RESULTS OF PROTEIN DATA SET

1NFN	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.939605	-0.771428571	0.579408565	1	1	0	0	0
#1 MCCE	1.272241	-0.029	1.373820221	0	0	1	0	0
#2 MCCE	0.793807	-0.297	0.795135837	1	0	0	0	0
#3 GA	1.114665	-0.102857143	1.19883922	1	0	1	0	0
#4 GA	0.948287	-0.185285714	1.004524716	1	1	0	0	0
#5 GA	1.108512	-0.091714286	1.193224443	1	0	1	0	0
#6 GA	0.974671	-0.174571429	1.035740936	1	1	0	0	0
#7 GA	0.667572	-0.565857143	0.38257959	1	0	0	0	0
#8 GA	0.657137	-0.522142857	0.430961881	1	0	0	0	0

Table B.3: Protein 1NFN

1GS9	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.661168	-0.542857143	0.407664661	2	0	0	0	0
Published	0.81626	-0.614285714	0.58059358	0	1	0	0	0
#1 MCCE	0.570692	-0.240714286	0.558901218	1	0	0	0	0
#2 MCCE	0.713537	-0.443571429	0.603690555	1	0	0	0	0
#3 GA	0.5111	-0.273857143	0.466115304	1	0	0	0	0
#4 GA	0.717649	-0.428	0.622206825	0	1	0	0	0
#5 GA	0.522493	-0.273571429	0.480815577	1	0	0	0	0
#6 GA	0.674974	-0.314	0.645362947	1	0	0	0	0
#7 GA	0.5858	-0.352	0.505767733	1	0	0	0	0

Table B.4: Protein 1GS9

1IGD	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.623431	-0.433333333	0.463937598	2	0	0	0	2
#1 MCCE	0.479494	-0.116	0.481580731	0	0	0	0	2
#2 MCCE	0.547301	-0.2874	0.48211525	1	0	0	0	2
#3 GA	0.497005	-0.064066667	0.510156764	0	0	0	0	2
#4 GA	0.456653	-0.104866667	0.460047958	0	0	0	0	2
#5 GA	0.479343	-0.123866667	0.479315266	1	0	0	0	2
#6 GA	0.455429	-0.127133333	0.452673466	0	0	0	0	2
#7 GA	0.702174	-0.3728	0.615921516	2	1	0	0	2
#8 GA	0.619131	-0.3104	0.55450284	2	0	0	0	2

Table B.5: Protein 1IGD

APPENDIX B - STATISTICAL RESULTS OF PROTEIN DATA SET

4PTI	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.560886	-0.192142857	0.546839571	2	0	0	0	0
Published	0.69213	-0.142071429	0.702963011	1	0	0	0	0
#1 MCCE	0.656037	-0.214357143	0.643433651	2	0	0	0	0
#2 MCCE	0.7955	-0.217285714	0.794136922	3	0	0	0	0
#3 GA	0.718669	-0.1575	0.727667374	2	0	0	0	0
#4 GA	0.780989	-0.111642857	0.802146889	3	0	0	0	0
#5 GA	0.755777	-0.178714286	0.762063537	3	0	0	0	0
#6 GA	0.910895	-0.286357143	0.897355311	4	1	0	0	0
#7 GA	0.801563	-0.282714286	0.7783646	2	1	0	0	0

Table B.6: Protein 4PTI

1IG5	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.620034	-0.072631579	0.632638585	0	1	0	0	0
#1 MCCE	0.492216	-0.009526316	0.505608695	1	0	0	0	0
#2 MCCE	0.724194	-0.154315789	0.726950713	0	0	1	0	0
#3 GA	0.647028	-0.024789474	0.664270325	0	1	0	0	0
#4 GA	0.621772	-0.032631579	0.637929482	2	0	0	0	0
#5 GA	0.630618	-0.027052632	0.647302134	0	1	0	0	0
#6 GA	0.624918	-0.047	0.640224179	2	0	0	0	0
#7 GA	0.571396	0.024368421	0.58651913	2	0	0	0	0
#8 GA	0.787096	-0.024368421	0.808277057	1	0	1	0	0

Table B.7: Protein 1IG5

4LZT	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.723763	-0.39	0.627375486	2	1	0	0	1
#1 MCCE	0.79606	-0.360333333	0.730418816	3	1	0	0	1
#2 MCCE	0.645327	-0.243944444	0.61476351	3	0	0	0	1
#3 GA	0.817624	-0.342833333	0.763796112	2	2	0	0	1
#4 GA	0.854503	-0.358277778	0.798256805	2	2	0	0	1
#5 GA	0.801882	-0.359277778	0.737675268	2	2	0	0	1
#6 GA	0.815984	-0.299722222	0.780947449	2	2	0	0	1
#7 GA	0.653221	-0.128222222	0.659082031	2	0	0	0	1
#8 GA	0.688042	-0.280722222	0.64638113	0	1	0	0	1

Table B.8: Protein 4LZT

APPENDIX B - STATISTICAL RESULTS OF PROTEIN DATA SET

1DWR	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.811028	0.396666667	0.774923652	0	1	0	0	0
#1 MCCE	0.778873	0.312166667	0.781686489	1	0	0	0	0
#2 MCCE	0.852454	0.5355	0.726567478	1	1	0	0	0
#3 GA	0.880272	0.4205	0.847154472	0	1	0	0	0
#4 GA	0.937277	0.490666667	0.87480413	1	1	0	0	0
#5 GA	0.852048	0.44	0.799288934	0	1	0	0	0
#6 GA	0.937046	0.499833333	0.868254206	1	1	0	0	0
#7 GA	0.891494	0.411333333	0.866417836	0	1	0	0	0
#8 GA	0.874496	0.490166667	0.793331562	0	1	0	0	0

Table B.9: Protein 1DWR

1A6K	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.848528	-0.257142857	0.772133717	2	0	0	0	4
#1 MCCE	0.934071	-0.052571429	0.831147171	2	0	0	0	4
#2 MCCE	1.003443	-0.200571429	0.921487721	3	0	0	0	4
#3 GA	0.855512	0.045285714	0.922763371	0	1	0	0	4
#4 GA	0.872189	0.092571429	0.936750208	0	1	0	0	4
#5 GA	0.877584	0.061	0.945605979	0	1	0	0	4
#6 GA	0.875626	0.083571429	0.941466561	0	1	0	0	4
#7 GA	0.853412	0.027571429	0.921308826	1	1	0	0	4
#8 GA	0.808263	0.030714286	0.872392823	0	1	0	0	4

Table B.10: Protein 1A6K

3RN3	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.897861	-0.076923077	0.93108704	1	2	0	0	2
#1 MCCE	1.051869	0.085692308	1.091180812	1	2	1	0	2
#2 MCCE	0.86809	-0.117461538	0.89522703	1	2	0	0	2
#3 GA	1.158011	0.473923077	1.099736064	3	0	0	1	2
#4 GA	1.137897	0.478692308	1.074461988	1	2	1	0	2
#5 GA	1.151903	0.474307692	1.092584046	3	0	0	1	2
#6 GA	1.116457	0.491769231	1.043244311	1	2	1	0	2
#7 GA	1.277976	0.078692308	1.327635705	1	1	1	1	2
#8 GA	1.334818	-0.266	1.361457001	0	0	2	1	2

Table B.11: Protein 3RN3

APPENDIX B - STATISTICAL RESULTS OF PROTEIN DATA SET

1GOA	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.965266	-0.073913043	0.984062725	4	1	2	0	2
#1 MCCE	0.968788	-0.135869565	0.980770861	4	2	1	0	2
#2 MCCE	1.067462	-0.052608696	1.090126504	2	2	2	0	2
#3 GA	1.03971	-0.049434783	1.061874536	3	4	1	0	2
#4 GA	1.021765	-0.184434783	1.027568349	1	3	2	0	2
#5 GA	1.037407	-0.025956522	1.0603903	2	4	1	0	2
#6 GA	0.948661	-0.123391304	0.961742251	0	4	1	0	2
#7 GA	1.053347	-0.032695652	1.07650138	5	0	1	1	2
#8 GA	0.896327	-0.209608696	0.891059775	2	1	1	0	2

Table B.12: Protein 1GOA

1RGG	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.850039	-0.266666667	0.830527826	0	3	0	0	6
#1 MCCE	0.902449	-0.280111111	0.882747841	3	1	1	0	6
#2 MCCE	0.976541	-0.371944444	0.92911118	4	1	1	0	6
#3 GA	1.021072	-0.281277778	1.010022644	2	1	2	0	6
#4 GA	0.975919	-0.266222222	0.966126137	1	1	2	0	6
#5 GA	1.00463	-0.265444444	0.997017979	1	1	2	0	6
#6 GA	0.965009	-0.243277778	0.960914503	1	1	2	0	6
#7 GA	0.933384	-0.339055556	0.894836723	0	1	2	0	6
#8 GA	0.946339	-0.313777778	0.918688807	0	2	1	0	6

Table B.13: Protein 1RGG

1I0V	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	1.28544	-0.427142857	1.258163671	4	0	0	1	0
#1 MCCE	1.388518	-0.531	1.331404868	0	0	0	1	0
#2 MCCE	1.150557	-0.437071429	1.104484249	4	2	1	0	0
#3 GA	1.422556	-0.349214286	1.431083354	0	0	1	1	0
#4 GA	1.423222	-0.3685	1.426581646	0	0	1	1	0
#5 GA	1.388366	-0.308357143	1.404789807	0	0	1	1	0
#6 GA	1.374077	-0.325071429	1.385469066	0	0	1	1	0
#7 GA	1.112947	-0.220142857	1.132139626	2	0	2	0	0
#8 GA	0.863019	-0.256285714	0.855195743	3	1	0	0	0

Table B.14: Protein 1I0V

APPENDIX B - STATISTICAL RESULTS OF PROTEIN DATA SET

1DG9	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	1.711724	1.7	0.282842712	1	1	0	0	0
#1 MCCE	2.27301	2.2645	0.277892965	0	0	2	2	0
#2 MCCE	2.705999	2.704	0.14707821	0	0	2	2	0
#3 GA	1.866797	1.857	0.27011479	0	1	1	1	0
#4 GA	2.012026	2.005	0.237587878	0	1	1	1	0
#5 GA	1.874758	1.866	0.255972655	0	1	1	1	0
#6 GA	2.034246	2.029	0.20647518	0	1	1	1	0
#7 GA	1.948221	1.9285	0.39103005	0	1	1	1	0
#8 GA	1.855594	1.845	0.280014285	0	1	1	1	0

Table B.15: Protein 1DG9

1H4G	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	1.267183	0.628666667	1.138858242	4	0	3	0	4
#1 MCCE	1.804905	0.142133333	1.862452372	2	1	2	2	4
#2 MCCE	1.68992	0.1918	1.737930099	1	1	2	2	4
#3 GA	1.987941	0.2056	2.046679569	2	1	2	3	4
#4 GA	2.002475	0.1862	2.063778719	2	2	1	3	4
#5 GA	1.972961	0.232733333	2.027950565	2	1	2	3	4
#6 GA	1.947026	0.220066667	2.002448946	1	1	2	3	4
#7 GA	1.930096	0.267733333	1.976524395	1	1	3	2	4
#8 GA	2.005857	0.180333333	2.067851427	1	1	2	3	4

Table B.16: Protein 1H4G

1XNB	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.855862	-0.225	0.882771609	1	1	0	0	1
#1 MCCE	1.234048	-0.310375	1.276845212	2	2	1	0	1
#2 MCCE	0.842928	-0.098875	0.894907567	2	0	0	0	1
#3 GA	0.93374	-0.07325	0.995133839	1	1	0	0	1
#4 GA	0.940791	-0.11975	0.997566861	2	1	0	0	1
#5 GA	0.977001	-0.18075	1.026427612	1	1	0	0	1
#6 GA	0.863592	-0.018875	0.922998133	2	0	0	0	1
#7 GA	0.962422	0.02675	1.028474703	3	1	0	0	1
#8 GA	1.022219	-0.076375	1.089743474	1	0	1	0	1

Table B.17: Protein 1XNB

APPENDIX B - STATISTICAL RESULTS OF PROTEIN DATA SET

1KXI	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.403113	0.025	0.464578662	0	0	0	0	0
#1 MCCE	0.164325	0.04975	0.180841321	0	0	0	0	0
#2 MCCE	0.336071	-0.31525	0.134467778	0	0	0	0	0
#3 GA	0.571284	0.01125	0.659534369	0	0	0	0	0
#4 GA	0.379559	0.02975	0.436928198	0	0	0	0	0
#5 GA	0.395073	-0.1225	0.433707659	0	0	0	0	0
#6 GA	0.501978	0.0515	0.576575234	0	0	0	0	0
#7 GA	0.661302	0.3385	0.655986026	1	0	0	0	0
#8 GA	0.471704	-0.1125	0.528959671	0	0	0	0	0

Table B.18: 1KXI

2CI2	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.774238	0.233333333	0.783022988	3	0	0	0	0
#1 MCCE	0.769762	0.214888889	0.783996244	2	0	0	0	0
#2 MCCE	0.4777	0.169333333	0.473776319	0	0	0	0	0
#3 GA	1.004921	0.595333333	0.858705421	0	2	0	0	0
#4 GA	0.917544	0.591	0.744433509	1	1	0	0	0
#5 GA	1.006836	0.599777778	0.85774862	0	2	0	0	0
#6 GA	0.953387	0.638777778	0.75068132	1	1	0	0	0
#7 GA	0.993052	0.520222222	0.897196157	1	0	1	0	0
#8 GA	0.864514	0.427	0.797299505	2	1	0	0	0

Table B.19: 2CI2

2CPL	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	1.760682	1.133333333	1.650252506	1	0	1	0	1
#1 MCCE	2.194066	1.338	2.129676971	1	0	0	1	1
#2 MCCE	1.862147	1.133666667	1.809305484	1	0	1	0	1
#3 GA	1.548454	0.904333333	1.539426625	0	0	1	0	1
#4 GA	1.660235	1.008333333	1.615383649	1	0	1	0	1
#5 GA	1.551163	0.912333333	1.536435268	0	0	1	0	1
#6 GA	1.655519	1.001	1.614965325	1	0	1	0	1
#7 GA	1.774928	1.073	1.731635354	1	0	1	0	1
#8 GA	1.860971	1.068333333	1.866230782	0	0	1	0	1

Table B.20: Protein 2CPL

APPENDIX B - STATISTICAL RESULTS OF PROTEIN DATA SET

1HNG	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.817256	-0.333571429	0.774245113	2	0	1	0	0
#1 MCCE	0.658652	-0.137285714	0.668503314	3	0	0	0	0
#2 MCCE	0.658514	-0.168785714	0.660542921	3	0	0	0	0
#3 GA	0.775704	0.152571429	0.789261945	1	1	0	0	0
#4 GA	**0.702934**	**0.1585**	**0.710683501**	**1**	**1**	**0**	**0**	**0**
#5 GA	0.807527	0.181214286	0.816637588	1	1	0	0	0
#6 GA	0.738824	0.198571429	0.73850303	1	1	0	0	0
#7 GA	0.901745	0.217428571	0.908174646	4	1	0	0	0
#8 GA	0.794165	0.072428571	0.820709516	1	1	0	0	0

Table B.21: Protein 1HNG

3EBX	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	0.2	0.2	-	-	-	-	-	2
#1 MCCE	0.581	0.581	-	-	-	-	-	2
#2 MCCE	0.023	0.023	-	-	-	-	-	2
#3 GA	0.293	0.293	-	-	-	-	-	2
#4 GA	0.294	0.294	-	-	-	-	-	2
#5 GA	0.284	0.284	-	-	-	-	-	2
#6 GA	**0.259**	**0.259**	-	-	-	-	-	**2**
#7 GA	0.483	0.483	-	-	-	-	-	2
#8 GA	0.425	0.425	-	-	-	-	-	2

Table B.22: Protein 3EBX - Out of 3 residues, 2 were out of range

1BEO	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	1.263826	-0.78125	1.149608629	3	1	1	0	3
#1 MCCE	1.231999	-0.726375	1.149064837	3	1	1	0	3
#2 MCCE	1.334338	-0.617625	1.326384476	1	2	1	0	3
#3 GA	1.174505	-0.8885	0.971007695	4	1	0	0	3
#4 GA	1.168369	-1.032625	0.830314549	2	1	1	0	3
#5 GA	**1.131652**	**-0.874625**	**0.919848357**	**3**	**1**	**0**	**0**	**3**
#6 GA	1.16218	-1.076875	0.774823923	2	1	1	0	3
#7 GA	1.262517	-0.771875	1.143928866	2	1	1	0	3
#8 GA	1.2247	-0.846125	1.034959352	3	0	1	0	3

Table B.23: Protein 1BEO

APPENDIX B - STATISTICAL RESULTS OF PROTEIN DATA SET

1PPF	RMSD	AVERAGE DIFFERENCE	DIFFERENCE STD.DEV.	# Errors 1-1.5	# Errors 1.5-2.0	# Errors 2.0-3.0	# Errors >3.0	Out of Range
Published	1.245994	0.558333333	1.163426236	1	2	1	0	2
#1 MCCE	1.252027	0.544833333	1.177390476	2	0	2	0	2
#2 MCCE	1.195389	0.54	1.113890153	3	0	2	0	2
#3 GA	1.232807	0.496	1.17881142	3	0	2	0	2
#4 GA	1.239403	0.513333333	1.17826192	3	0	2	0	2
#5 GA	1.239258	0.492833333	1.187606825	3	0	2	0	2
#6 GA	1.237668	0.531916667	1.167227909	3	0	2	0	2
#7 GA	1.18277	0.640916667	1.038269492	1	0	1	1	2
#8 GA	1.228948	0.682666667	1.067339116	2	1	0	1	2

Table B.24: Protein 1PPF

Appendix C

MCCE and Hybrid GA/SA Parameters

```
4.0       Protein dielectric constant for DelPhi                (EPSILON_PROT)

ph        "ph" for pH titration, "eh" for eh titration          (TITR_TYPE)
0.0       Initial pH                                            (TITR_PH0)
0.5       pH interval                                           (TITR_PHD)
0.0       Initial Eh                                            (TITR_EH0)
30.0      Eh interval (in mV)                                   (TITR_EHD)
29        Number of titration points                            (TITR_STEPS)

f        Minimize output files                                  (MINIMIZE_SIZE)

Options specific to each step:

step 1:
f         Label terminal residues?                              (TERMINALS)
2.0       distance limit of reporting clashes                   (CLASH_DISTANCE)
0.05      cut off water if % SAS exceeds this number            (H2O_SASCUT-OFF)

step 2:
f         Use specific control (head1.lst) on rotamer making    (ROT_SPECIF)
0         Numble of cycles used for initial minimization        (N_INITIAL_RELAX)
f         Rebuild sidechain based on torsion minima             (REBUILD_SC)

t         Do swap (stereo isotope)                              (ROT_SWAP)
t         Do rotate?                                            (PACK)
12        number of rotamers in a bond rotation                 (ROTATIONS)
0.25      SAS cuttoff of making fewer rotamers                  (SAS_CUT-OFF)
10.0      Cut-off of self vdw in kcal/mol                        (VDW_CUT-OFF)
5000      number of repacks                                     (REPACKS)
0.05      occupancy cut-off of repacks                           (REPACK_CUT-OFF)

t         h-bond directed rotamer making, requir 't' on Do rotate  (HDIRECTED)
1.0       threshold for two conformers being different          (HDIRDIFF)
36        Limit number of the Hbond conformers                  (HDIRLIMT)

3         number of cycles to relax rotamer clash               (HV_RELAX_NCYCLE)
1         time (in fs) for each relaxation iteration            (HV_RELAX_DT)
100       number of iterations                                  (HV_RELAX_NITER)
2.        relax rotamer if vdw interaction is bigger than this (HV_RELAX_VDW_THR)
5.        not relax rotamer if heavy atom vdw is bigger than this (HV_RELAX_HV_VDW_THR)
20.       scaling factor for torsion during relaxation          (HV_TORS_SCALE)
10000     maximun n steps of shake                              (HV_RELAX_N_SHAKE)
1.0       constraint distance                                   (HV_RELAX_CONSTRAINT)
20.       constraint force for atom stay in original position (HV_RELAX_CONSTRAINT_FRC)
-3.0      relax if electrostatic interaction is more favorable than this, and the charged
groups are close  (HV_RELAX_ELEC_THR)
```

APPENDIX C - MCCE AND HYBRID GA/SA PARAMETERS

```
0.3      threshold used to define a charged atom, see hv_relax_elec_dist_thr
(HV_RELAX_ELEC_CRG_THR)
3.0      only relax electrostatically favorable pairs that has charged atom distance
shorter than this (HV_RELAX_ELEC_DIST_THR)

t        Do relaxation on hydrogens                                  (RELAX_H)
-1.0     Energy threshold for keeping the conformer                  (RELAX_E_THR)
100      Loop over N local microstates                               (RELAX_NSTATES)
12       default number of hydroxyl positions                        (RELAX_N_HYD)
5.       do not relax hydrogen if vdw bwt two sidechain conformer bigger than this
(RELAX_CLASH_THR)
1.0      phi for each step of relaxation                             (RELAX_PHI)
300      Maximum number of steps of relaxation                       (RELAX_NITER)
0.5      Torque threshold for keep relaxing                          (RELAX_TORQ_THR)

step 3:
80.0     Solvent dielectric constant for DelPhi                      (EPSILON_SOLV)
65       Grids in each DelPhi                                        (GRIDS_DELPHI)
2.0      The target grids per angstrom for DelPhi                    (GRIDS_PER_ANG)
1.4      Radius of the probe                                         (RADIUS_PROBE)
2.0      Ion radius                                                  (IONRAD)
0.15     Salt                                                        (SALT)

step 4:
5.0      Big pairwise threshold to make big list                     (BIG_PAIRWISE)
-1       Random seed, -1 uses time as random seed                    (MONTE_SEED)
298.15   Temperature                                                 (MONTE_T)
3        Number of flips                                             (MONTE_FLIPS)
100      Annealing = n_start * confs                                 (MONTE_NSTART)
300      Equalibration = n_eq * confs                                (MONTE_NEQ)
0.001    Cut-off occupancy of the reduction                          (MONTE_REDUCE)
30       Independent monte carlo sampling                            (MONTE_RUNS)
2000     Sampling = n_iter * confs                                   (MONTE_NITER)
50000    Trace energy each MONTE_TRACE steps, 0 no trace             (MONTE_TRACE)
1000000  Maximum microstates for analytical solution                 (NSTATE_MAX)
t        Do entropy correction                                       (MONTE_TSX)

# Advanced options:
1         delphi start conformer number, 0 based                     (DELPHI_START)
99999     delphi end conformer number, self included                 (DELPHI_END)
/tmp      delphi temporary file folder, "/tmp" uses node             (DELPHI_FOLDER)
t         clean up delphi focusing directory?                        (DELPHI_CLEAN)
debug.log additional error information                               (DEBUG_LOG)
new.tpl   local parameter file for unrecognized res                  (NEWTPL)
1.7      defalut van der Waals radius, for SAS                       (DEFAULT_RADIUS)
0.5      factor to 1-4 LJ potontial (1.0 is full)                    (FACTOR_14LJ)
6.0      dielectric constant for Coulomb's law                       (EPSILON_COULOMB)

2.0      Pruning cut-off of RMSD                                     (PRUNE_RMSD)
1.5      Pruning cut-off of eletrostatic pairwise                    (PRUNE_ELE)
1.5      Pruning cut-off of vdw pairwise                             (PRUNE_VDW)
-0.06    coefficient used to calculate vdw interaction with implicit water (SAS2VDW)

############GENETIC ALGORITHM PARAMETERS#################
3000     Population size of each GA                                  (GA_POP_SIZE)
0.1      Mutation rate for each GA                                   (GA_MUTATION_RATE)
0.0      Migration rate for each GA                                  (GA_MIGRATION_RATE)
0.9      Crossover rate for each GA                                  (GA_CROSSOVER_RATE)
30       Number of random cut points for crossover                   (GA_RANDOM_CUT_POINTS)
200      Random seed value, use -1 for random value                  (GA_SEED)
450      Maximum number of generation for each GA                    (GA_GENERATIONS)
10,15    Distribution centre for the evolutionary sampling           (GA_DIST_CENTRE)
2.0      Up to how many eps units from the min to include solns      (GA_DIST_CENTRE_EPSILON)
0.15     Overall population dE from the min                          (GA_CONVERGENCE_EPSILON)
300000   Max number of solns in the final bucket                     (GA_MAX_BUCKET_POP)
3.0,5.0  Residue lowest occupied energy cut-off                      (GA_LOCAL_RESIDUE_CUTOFF)
0,0.05,0.1 Occupancy conformers based filtering                      (GA_OCCUPANCY_CUTOFF)
0.7      Population percentage for dE calculation                    (GA_DELTA_E)
```