

Air Force Institute of Technology

AFIT Scholar

Theses and Dissertations

Student Graduate Works

3-7-2002

A Multiobjective Approach Applied to the Protein Structure Prediction Problem

Richard O. Day

Follow this and additional works at: <https://scholar.afit.edu/etd>



Part of the [Other Engineering Commons](#)

Recommended Citation

Day, Richard O., "A Multiobjective Approach Applied to the Protein Structure Prediction Problem" (2002).
Theses and Dissertations. 4447.
<https://scholar.afit.edu/etd/4447>

This Thesis is brought to you for free and open access by the Student Graduate Works at AFIT Scholar. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of AFIT Scholar. For more information, please contact AFIT.ENWL.Repository@us.af.mil.



**A MULTIOBJECTIVE APPROACH
APPLIED TO THE
PROTEIN STRUCTURE PREDICTION PROBLEM**

THESIS

Richard O. Day, Captain, USAF

AFIT/GE/ENG/02M-05

**DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY**

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

Approved for public release; distribution unlimited

Report Documentation Page

Report Date 7 Mar 02	Report Type Final	Dates Covered (from... to) Aug 2000 - Mar 2002
Title and Subtitle A Multiobjective Approach Applied to the Protein Structure Prediction Problem	Contract Number	
	Grant Number	
	Program Element Number	
Author(s) Capt Richard O. Day, USAF	Project Number	
	Task Number	
	Work Unit Number	
Performing Organization Name(s) and Address(es) Air Force Institute of Technology Graduate School of Engineering and Management (AFIT/EN) 2950 P Street, Bldg 640 WPAFB OH 45433-7765	Performing Organization Report Number AFIT/GE/ENG/02M-05	
Sponsoring/Monitoring Agency Name(s) and Address(es) AFRL Materials and Manufacturing Dir. Dr. Ruth Pachter WPAFB, OH 45433	Sponsor/Monitor's Acronym(s)	
	Sponsor/Monitor's Report Number(s)	
Distribution/Availability Statement Approved for public release, distribution unlimited		
Supplementary Notes The original document contains color images.		
Abstract Interest in discovering a methodology for solving the Protein Structure Prediction problem extends into many fields of study including biochemistry, medicine, biology, and numerous engineering and science disciplines. Experimental approaches, such as, x-ray crystallographic studies or solution Nuclear Magnetic Resonance Spectroscopy, to mathematical modeling, such as minimum energy models are used to solve this problem. Recently, Evolutionary Algorithm studies at the Air Force Institute of Technology include the following: Simple Genetic Algorithm (GA), messy GA , fast messy GA, and Linkage Learning GA, as approaches for potential protein energy minimization. Prepackaged software like GENOCOP, GENESIS, and mGA are in use to facilitate experimentation of these techniques. In addition to this software, a parallelized version of the fmGA, the so-called parallel fast messy GA, is found to be good at finding semi-optimal answers in reasonable wall clock time. The aim of this work is to apply a Multiobjective approach to solving this problem using a modified fast messy GA. By dividing the CHARMM energy model into separate objectives, it should be possible to find structural configurations of a protein that yield lower energy values and ultimately more correct conformations.		

Subject Terms fast messy Genetic Algorithm, Multiobjective, Protein Structure Prediction problem, High Performance Computing	
Report Classification unclassified	Classification of this page unclassified
Classification of Abstract unclassified	Limitation of Abstract UU
Number of Pages 228	

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

AFIT/GE/ENG/02M-05

A MULTIOBJECTIVE APPROACH APPLIED TO THE
PROTEIN STRUCTURE PREDICTION PROBLEM

THESIS

Presented to the Faculty of the School of Engineering and Management
of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Computer Engineering

Richard O. Day, B.S.C.E

Captain, USAF

March, 2002

Approved for public release; distribution unlimited

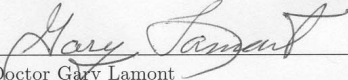
AFIT/GE/ENG/02M-05

A MULTIOBJECTIVE APPROACH APPLIED TO THE PROTEIN
STRUCTURE PREDICTION PROBLEM

Richard O. Day, B.S.C.E

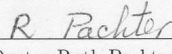
Captain, USAF

Approved:



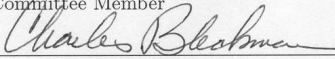
Doctor Gary Lamont
Thesis Advisor

6 March 02
Date



Doctor Ruth Pachter
Committee Member

8 March 02
Date



Doctor Charles Bleckmann
Committee Member

6 March 02
Date

Acknowledgements

This has been an eye opening experience for me. The struggle for researchers is hardening each day. The increased amount of knowledge needed in each field just for entrance in learning the background of a problem – let alone added credible work toward finding a solution to that problem – makes it extremely difficult for researchers such as myself. I give thanks to the following people who have helped me with this thesis: To my thesis advisor, Dr. Gary Lamont, who kept the whip cracking when I wanted to take a break and to my sponsor, Dr. Ruth Pachter, for without her I would not have been able to understand the biochemistry part of the problem. Additionally, I would like to acknowledge Dr. Charles Beckman for inspiring me to keep this thesis document concise.

Many other students have helped my understanding of the computer science topics and aided in publishing. Thanks to Jesse Zydallis and Steven Michaud for the nudging me toward publishing several papers. Also, thanks goes out to all my fellow students who provided support and timely distractions. Among these I would like to lend a special thanks to Eric Trias for his hand in keeping me fit and Todd Andel who constantly held the highest of standards helping me to aim my own higher still. Also I would like to thank the great administrative support and teachers within the engineering department for helping me countless times.

Richard Orison Day

Table of Contents

	Page
List of Figures	xii
List of Tables	xviii
List of Symbols	xxi
List of Abbreviations	xxii
Abstract	xxv
 I. Introduction	 1-1
1.1 Overview of PSP Problem	1-2
1.2 Research Goals	1-5
1.3 Assumptions	1-8
1.4 Sponsorship	1-9
1.5 Thesis Layout	1-10
 II. Problem Domain Models	 2-1
2.1 Problem Domain	2-2
2.1.1 Symbolic/Formalized Problem Domain Description	2-2
2.1.2 Genotype	2-5
2.1.3 Phenotype	2-5
2.2 Genotype and Phenotype of the PSP problem	2-6
2.2.1 Diagrams/Graphs for insight to problem domain	2-6
2.2.2 Simple problem vs. real-world problem instantiation	2-7

	Page
2.2.3 A Structural view point of the PSP problem .	2-11
2.3 Domain Formalization for cartesian coordinates	2-14
2.4 Importance of Energy Function	2-16
2.5 CHARMm ENERGY FUNCTION	2-17
2.6 Summary	2-20
III. Possible Algorithm Domains	3-1
3.1 Background	3-1
3.2 Experimental Methods	3-2
3.2.1 X-Ray Crystallography	3-2
3.2.2 Nuclear Magnetic Resonance Spectroscopy . .	3-3
3.3 Simulation Methods	3-4
3.3.1 Atomistic and Non-Atomistic Lattice Simulations	3-5
3.3.2 Molecular dynamic model simulation	3-8
3.3.3 Monte Carlo	3-9
3.3.4 Off-lattice Monte-Carlo Simulations	3-9
3.4 Energy Minimization Methods	3-10
3.5 Historical Perspective and AFIT GAs	3-11
3.5.1 Simple GA	3-12
3.5.2 Steady State GA	3-13
3.5.3 Messy GA	3-17
3.6 fast messy GA	3-20
3.7 Parallel fast messy GA	3-24
3.8 Multi Objective fmGA (MOfmGA)	3-27
3.9 Basic evolutionary algorithmic approach justified . . .	3-27
3.10 Simplified Mapping Problem Domain to Algorithm Do- main	3-28
3.10.1 Mathematical/Symbolic model	3-28

	Page
3.11 Problem/Algorithm Domain Operational Design Specification form	3-29
3.11.1 Extended Algorithm Domain	3-30
3.12 Mapping Problem Domain to Algorithm Domain DFS/BT	3-31
3.13 Mapping Problem Domain to Algorithm Domain Stochastic	3-31
3.14 Comparison between mappings	3-33
3.15 Summary	3-33
IV. Development Methodology	4-1
4.1 Algorithm Design	4-1
4.2 Research Design	4-2
4.2.1 Multiple Competitive Templates	4-2
4.2.2 Farming Model	4-8
4.2.3 Building Block Analysis	4-10
4.2.4 Protein 3D File Generation	4-10
4.2.5 Multiple Objective	4-11
4.2.6 Ramachandran	4-16
4.2.7 RMS difference	4-16
4.3 Software Engineering Approaches	4-17
4.4 Summary	4-18
V. Design of Experiments	5-1
5.1 Justification of experimental design	5-1
5.2 System Under Test	5-3
5.3 Component Under Study	5-3
5.3.1 Effectiveness Components	5-4
5.3.2 Efficiency Component	5-4
5.4 System Services	5-4

	Page
5.5 Design Discussion	5-5
5.6 Performance Metric	5-7
5.7 Parameters	5-7
5.7.1 System Parameters	5-7
5.7.2 Workload Parameters	5-10
5.8 Algorithm Factors	5-11
5.9 Hypothesis	5-12
5.10 Evaluation Techniques	5-12
5.11 Workload Selection	5-12
5.12 Experimental Design	5-13
5.13 Analyze and Interpret results	5-14
5.14 Statistical Techniques	5-15
5.14.1 Kruskal-Wallis	5-15
5.14.2 t-test Paired/Unpaired Observations	5-15
5.15 Presentation Techniques	5-16
5.16 Summary	5-16
VI. Results and Analysis	6-1
6.1 Multiple Competitive Templates	6-1
6.1.1 Results for Multiple Competitive Template Ex- periment with MET	6-1
6.1.2 Results for Multiple Competitive Template Ex- periment with POLY	6-3
6.2 Farming model experiment	6-6
6.2.1 Farming Experiment	6-8
6.3 Building Block Size analysis	6-11
6.4 Protein 3D File Generation	6-13
6.5 Multiobjective Experiment	6-14

	Page
6.6 Ramachandran Experiment	6-17
6.7 RMS difference	6-23
6.8 Comparing to other Research	6-25
6.9 Summary	6-26
VII. Conclusions and Recommendations	7-1
7.1 Recommendations	7-2
7.1.1 Summary	7-2
Appendix A. Chemical Formulas for Amino Acids	A-1
Appendix B. 2D Conformation/Chemical Formulation for Amino Acids	B-1
Appendix C. Ramachandran Worksheets	C-1
C.1 Alanine	C-1
C.2 Arginine	C-2
C.3 Asparagine	C-3
C.4 Aspartic acid	C-4
C.5 Cysteine	C-5
C.6 Glutamine	C-6
C.7 Glutamic	C-7
C.8 Glycine	C-8
C.9 Histidine	C-9
C.10 Isoleucine	C-10
C.11 Leucine	C-11
C.12 Lysine	C-12
C.13 Methionine	C-13
C.14 Phenylalanine	C-14
C.15 Proline	C-15

	Page
C.16 Serine	C-16
C.17 Threonine	C-17
C.18 Tryptophan	C-18
C.19 Tyrosine	C-19
C.20 Valine	C-20
Appendix D. Computational Platforms	D-1
Appendix E. fmGA Configuration File	E-1
Appendix F. Biological Relations to GA Operators	F-1
F.1 Biology Background	F-1
F.1.1 Reproduction	F-1
F.1.2 Competition	F-1
F.1.3 Selection	F-1
F.1.4 Crossover	F-2
F.1.5 Mutation	F-2
F.1.6 Transposition	F-3
F.1.7 Translocation	F-3
F.1.8 Conjugation	F-3
F.1.9 Inversion	F-4
F.1.10 Transduction	F-4
F.1.11 Gametogenesis	F-4
F.1.12 Transcription	F-4
F.1.13 Translation	F-4
F.1.14 Exons	F-4
F.1.15 Introns	F-4
Appendix G. Building Blocks	G-1

	Page
Appendix H. Other General AFIT Approaches	H-1
H.1 Combined Algorithms (local search)	H-1
H.2 Hybridized GA	H-2
H.3 Real Valued GA	H-2
H.4 Linkage Learning GA	H-3
Appendix I. Basic Evolutionary Algorithmic Approaches Justified .	I-1
I.1 Evolutionary Strategies	I-1
I.1.1 Mutation	I-1
I.1.2 Recombination	I-1
I.1.3 Selection	I-2
I.1.4 <i>Application</i>	I-2
I.2 Evolutionary Programming	I-3
I.2.1 Mutation	I-3
I.2.2 Recombination	I-3
I.2.3 Selection	I-3
I.2.4 <i>Application</i>	I-3
I.3 Genetic Algorithms (GA)	I-4
I.3.1 fast messy Genetic Algorithm (fmGA)	I-4
I.3.2 Application	I-4
I.4 Genetic Programming	I-5
I.4.1 Description	I-5
Appendix J. Data and task decomposition design	J-1
J.1 Multi Objective fmGA Data and Task decomposition .	J-1
J.1.1 Design of operators and parameter values . . .	J-3
J.1.2 Implementation	J-5

	Page
Appendix K. Multiobjective Discussion	K-1
K.1 Multiobjective Optimization	K-1
K.1.1 Pareto Terminology	K-2
Appendix L. Data from multiple competitive template experiments on MET	L-1
Appendix M. Farming Model Experiment Graphs and Table	M-1
Appendix N. Atom and Amino Acid Identification	N-1
Bibliography	BIB-1

List of Figures

Figure		Page
1.1.	Description of the protein creation and folding process [15] .	1-3
1.2.	A few principles of protein folding [15]	1-4
2.1.	2D matrix representing the allowable angles for each set up 3 atoms defined by Figure 2.8.	2-3
2.2.	ϑ is shown to have the invalid bit-degrees angle of one. This supports the need to identify these angles as being unrepresentable within a conformation of a protein.	2-4
2.3.	Illustration of Genotype to Phenotype domain	2-5
2.4.	High Level abstraction of a protein consisting of n amino acids.	2-7
2.5.	Un-ionized form of an amino acid	2-7
2.6.	Representation of Protein revealing atoms bonded together to make the linear chain of Amino Acids	2-7
2.7.	Simple three atom model	2-8
2.8.	Simple three atom discretized model	2-9
2.9.	Fictitious protein	2-10
2.10.	ϕ angle representation.	2-11
2.11.	ψ angle representation	2-12
2.12.	ω angle representation	2-13
2.13.	χ angle representation	2-13
2.14.	Simple example of four atoms making a measurable dihedral angle.	2-15
2.15.	Graphical description of energy functions and how they are translated from physical atom-bond relationships to Potential Energy Functions.	2-16

Figure		Page
3.1.	X-ray Diffraction Equipment found at Lawrence Livermore National Laboratory [88]	3-3
3.2.	Year vs Total Available Structures held in the Protein Data Bank [26].	3-4
3.3.	An example of using a lattice space model (<i>b</i>) to represent atoms in crystal form (<i>a</i>) [91].	3-6
3.4.	An example crystalized graphite growth [91].	3-6
3.5.	Six grid alternative selection configurations are shown: a) multi-block structured, b) unstructured-triangular, c) unstructured-quadrilateral, d) gybrid, e) Chimera, and f) hierarchical grid [81].	3-8
3.6.	Illustrating the density boxes and analysis of atoms within each box during the Off-Lattice Simulation method [60].	3-10
3.7.	High Level overview of solving real world problem with a sGA.	3-14
3.8.	An example of single point crossover applied after selecting two population members.	3-16
3.9.	A simple example of mutation applied.	3-17
3.10.	Program flow of the mGA [70]. The complexity for this algorithm can be found in Table 3.5	3-19
3.11.	Program flow of the fmGA.	3-21
3.12.	An example of typical calculations to find population sizes for the fmGA and mGA. It should be noted that the population size change for the fmGA as the building block size change throughout the algorithm.	3-24
3.13.	The upper plot is of the fmGA calculated population sizes and the lower plot is that of mGA population sizes. The fmGA consistently produces population sizes orders of magnitude lower than that of the mGA. Population sizes are on the z axis while the x and y are reflecting indexes to building block sizes of 15-75 and string lengths of 20-80.	3-25

Figure		Page
4.1.	Farming model visualization of communication between algorithm nodes and farm or compute nodes. Algorithm nodes are represented by the square boxes and Compute nodes are the Ovals.	4-8
4.2.	Visualization of nodes grouped into Algorithm and Farm arrays.	4-10
4.3.	An example of how the mapping of the Ramachandran plots works in with the algorithm. For each evaluation, the mapping must be accomplished for each dihedral interpretation. This is quite computationally expensive.	4-17
6.1.	Building Block Test vs. Fitness plot of results for an experiment using multiple methods of competitive template generation on the protein MET.	6-3
6.2.	Summary of results after conducting the paired observation difference test found in [47] on the multiple competitive template experiment with MET.	6-3
6.3.	Building Block Test vs. Time to Complete plot of results for an experiment using multiple methods of competitive template generation on the protein MET.	6-4
6.4.	Building Block Test vs. Fitness plot of results for an experiment using multiple methods of competitive template generation on the protein POLY.	6-7
6.5.	Summary of results after conducting the paired observation difference test found in [47] on the multiple competitive template experiment with POLY.	6-7
6.6.	Building Block Test vs. Time to Complete plot of results for an experiment using multiple methods of competitive template generation on the protein POLY.	6-8
6.7.	Time vs. BB Test for 1 Algorithm Node. Validation of fitness values before and after using the Farming model can be found in Appendix M	6-9

Figure	Page
6.8. Time vs. BB Test for 2 Algorithm Nodes. Validation of fitness values before and after using the Farming model can be found in Appendix M	6-10
6.9. Time vs. Building Block Test plot of building block sizes and associated best fitness found from each experiment. Specific computers used in this experiment can be found in Appendix M in Table M.1	6-12
6.10. Conformation from a PDB file [17]. This protein had a fitness value of -33 kcal/mol.	6-13
6.11. Conformation from a PDB file [17]. This protein had a fitness value of -169 kcal/mol.	6-13
6.12. Conformation from a PDB file representing the accepted conformation for <i>Polyalanine</i> ₁₆ . Notice that the conformation is nearly the same found in our MOfmGA search (represented in Figure 6.11).	6-14
6.13. [Met]-enkephlan Pareto Front.	6-15
6.14. <i>Polyalanine</i> ₁₄ Pareto Front	6-15
6.15. Building Block Test vs. Fitness plot of results for an experiment using no, pesimistic and optimistic Ramachandran plots on the protein POLY. See Appendix C for the restrictions applied to the landscape for each different method.	6-18
6.16. Summary of results after conducting the paired observation difference test found in [47] on the experiment using no, pessimistic and optimistic Ramachandran plots on the protein POLY. . .	6-18
6.17. Summary of results after conducting the Kruskal-Wallis test on the results from the pessimistic and optimistic implementation of the Ramachandran Plots. Note: the test concludes 91% confidence that these are different; furthermore, the pessimistic constraints more effective.	6-19
6.18. Building Block Test vs. Time to Complete plot of results for an experiment using no, pesimistic and optimistic Ramachandran plots on the protein POLY.	6-19

Figure		Page
6.19.	Generation of a Randomly created structure vs. Fitness . . .	6-20
6.20.	$\ln(\text{Fitness})$ vs. Frequency Found (Randomly generated conformations)	6-20
6.21.	RMS Cartesian Coordinate difference for all atoms vs. RMS Cartesian Coordinate difference for only backbone atoms. This is for validations of the RMS calculation and to ensure that we have a linearity between these two differences. Both RMS differences are measured in Angstroms (\AA). Where $1 \text{ \AA} = 10^{-10}$	6-21
6.22.	Fitness vs. RMS Cartesian Coordinate difference (kcal/mol vs. Angstroms (\AA)). (Fitness evaluation on randomly generated conformations)	6-22
6.23.	Fitness vs. RMS Dihedral Angle difference. (kcal/mol vs. Radians) (Fitness evaluation on randomly generated conformations)	6-22
A.1.	List of linear structure formula for Amino Acids. [30]	A-1
B.1.	Conformation/Chemical formulation for Amino Acids.	B-1
C.1.	Alanine Ramachandran Worksheet.	C-1
C.2.	Arginine Ramachandran Worksheet.	C-2
C.3.	Asparagine Ramachandran Worksheet.	C-3
C.4.	Aspartic acid Ramachandran Worksheet.	C-4
C.5.	Cysteine Ramachandran Worksheet.	C-5
C.6.	Glutamine Ramachandran Worksheet.	C-6
C.7.	Glutamic Ramachandran Worksheet.	C-7
C.8.	Glycine Ramachandran Worksheet.	C-8
C.9.	Histidine Ramachandran Worksheet.	C-9
C.10.	Isoleucine Ramachandran Worksheet.	C-10
C.11.	Leucine Ramachandran Worksheet.	C-11
C.12.	Lysine Ramachandran Worksheet.	C-12
C.13.	Methionine Ramachandran Worksheet.	C-13

Figure		Page
C.14.	Phenylalanine Ramachandran Worksheet.	C-14
C.15.	Proline Ramachandran Worksheet.	C-15
C.16.	Serine Ramachandran Worksheet.	C-16
C.17.	Threonine Ramachandran Worksheet.	C-17
C.18.	Tryptophan Ramachandran Worksheet.	C-18
C.19.	Tyrosine Ramachandran Worksheet.	C-19
C.20.	Valine Ramachandran Worksheet.	C-20
D.1.	An illustration of the Pile of PCs network configuration. This configuration is best described as two separate crossbar switches; furthermore, it was uses as such during experimental runs in this Thesis.	D-2
D.2.	An illustration of how to cut the crossbar switches when calculating bisection width.	D-3
D.3.	An illustration of how to cut the myrnet's crossbar switches when calculating bisection width. Assuming that only one wire is cut to disconnect several processors from any one processor	D-4
D.4.	An illustration of a SP P3 omega network configuration. . . .	D-5
N.1.	MET's Amino Acid and Atom number identification figure. .	N-1
N.2.	PLOY's Amino Acid and Atom number identification figure. .	N-2

List of Tables

Table	Page
2.1. Comparison Of Common Energy Functions used in solving the PSP problem [96] [87] [20]	2-20
3.1. Classification of methodologies used in solving the PSP problem. For advantages and disadvantages see Sections 3.2, 3.3, and 3.4.	3-2
3.2. General algorithm for a Monte Carlo simulation.	3-9
3.3. Complexity Estimates for the sGA. Where l is the length of chromosome, n is the size of population, q is the group size for a tournament selection and g is the number of generations.	3-13
3.4. Complexity Estimates for the ssGA. Where l is the length of chromosome, n is the size of population and g is the number of generations of reproduction.	3-13
3.5. Complexity Estimates for the Original mGA [42]	3-18
3.6. Complexity Estimates for the fmGA [42]	3-20
3.7. Pseudo code for DFS/BT Algorithm	3-32
3.8. Comparison between Deterministic algorithm and Stochastic Algorithm applied to the PSP problem	3-33
4.1. Pseudo code for evaluation of partial solutions.	4-3
4.2. Competitive Template and Multiple Objective settings for the fmGA. This Table is describe more thoroughly in Appendix E.	4-4
4.3. Pseudo code for tournament selection.	4-5
4.4. Secondary Structures useful in matching onto polypeptides structures. Note 1 and 2 are used for competitive template generation. [12]	4-6
4.5. Pseudo code for finding and storing the best fitness.	4-6
4.6. Pseudo code for rotating competitive template mechanism.	4-7

Table		Page
4.7.	Pseudo code for evaluation of partial solutions.	4-9
4.8.	Pseudo code modifying the evaluation function when moving from the single to multiple objective code.	4-15
4.9.	Pseudo code for multiobjective tournament selection.	4-15
4.10.	List of Ramachandran plot variations.	4-16
5.1.	Options for the fmGA	5-6
5.2.	Parameters for design	5-7
5.3.	Options for the fmGA	5-8
5.4.	Input Schedule for the fmGA	5-9
5.5.	System and Workload Parameters	5-10
5.6.	List of possible workload parameters along with their associated search space.	5-13
6.1.	Time Complexity of Energy Minimization Methods [32]	6-9
6.2.	Best Fitness Found	6-17
6.3.	RMS difference calculations for POLY.	6-19
6.4.	RMS difference calculations for MET.	6-21
6.5.	Angles found for MET at ~ -38 kcal/mol.	6-23
6.6.	Angles found for POLY at ~ -170 kcal/mol.	6-24
D.1.	PPC cluster <i>Specifications</i>	D-1
D.2.	Summary of Pile of PCs (Switches combined, 2 Intel switches, 1 gigabit switch)	D-3
D.3.	Specifications for the Cluster of Workstations	D-3
D.4.	Summary of Throughput Potential for the Cluster of Workstations)	D-4
D.5.	Specifications for the Network of Workstations)	D-5
D.6.	SP P3 <i>Specifications</i>	D-5
D.7.	Summary of SP P3 Throughput Potential	D-6

Table		Page
M.1.	Computer Systems	M-1

List of Symbols

Symbol		Page
C_α	alpha-Carbon	1-3
N	Nitrogen atom	2-6
C	Carbon atom	2-6
ϕ	Dihedral angle made by the following atoms: N_i , $C_\alpha(i)$, $C_{(i)}$, and $N_{(i+1)}$	2-11
ψ	Dihedral angle made by the following atoms: $C_{\alpha(i)}$, $C_{(i)}$, $N_{(i+1)}$, and $C_{\alpha(i+1)}$	2-11
ω	Dihedral angle made by the following atoms: $C_{(i)}$, $N_{(i+1)}$, $C_{\alpha(i+1)}$, and $C_{(i+1)}$	2-11

List of Abbreviations

Abbreviation		Page
PSP	Protein Structure Prediction	xxv
NMR	Nuclear Magnetic Resonance	xxv
AFIT	Air Force Institute of Technology	xxv
GA	Genetic Algorithm	xxv
mga	messy GA	xxv
fmGA	fast messy GA	xxv
LLGA	Linkage Learning GA	xxv
pfmGA	parallel fast messy GA	xxv
MO	MultiObjective	xxv
MOfmGA	MutliObjective fast messy Genetic Algorithm	xxv
BB	Building Block	xxv
HGP	Human Genome Project	1-1
NHGRI	National Human Genome Research Institute	1-1
NIH	National Institute of Health	1-1
DOE	Department of Energy	1-1
MET	Met-Enkephlain	1-2
POLY	<i>Polyalanine</i> ₁₄	1-2
N	Nitrogen	1-3
C	Carbon	1-3
H	Hydrogen	1-3
mRNA	messenger RNA	1-4
tRNA	transfer RNA	1-4
EAs	Evolutionary Algorithms	1-7
ESs	Evolutionary Strategies	1-7
GP	Genetic Programming	1-7

Abbreviation		Page
EP	Evolutionary Programming	1-7
SA	Simulated Annealing	1-7
ANOVA	Analysis of Variance	1-8
HPC	high performance computing	1-9
AFRL	Air Force Research Laboratory	1-9
WPAFB	Wright Patterson Air Force Base	1-9
DOD	Department of Defense	1-9
NP	Nondeterministic Polynomial	2-1
PDB	protein data bank	2-2
WRT	with respect to	2-15
OPLS	Optimized Potentials Potentials for Liquid Simulations .	2-19
REM	Random Energy Model	2-19
LLNL	Lawrence Livermore National Laboratory	3-2
IBM	International Business Machines	3-5
flops	floating operations per second	3-5
MD	Molecular dynamic	3-8
AMBER	Assisted Model Building with Energy Refinement	3-11
ECEPP	Empirical Conformational Energy Program for Peptides	3-11
sGA	simple GA	3-11
PGA	Parallel GA	3-11
ssGA	Steady State GA	3-13
CT	competitive template	3-18
PCI	probabilistic complete initialization	3-22
BBF	building block filtering	3-22
DFS/BT	Dept First Search with backtracking	3-29
MPI	message passing interface	4-8
PDB	Protein Data Bank	4-10

Abbreviation		Page
VMD	Visual Molecular Dynamics	4-10
MOMGA-II	Multiobjective messy GA	4-11
RAD	radians	4-16
PPCs	Pile of PCs	5-3
COWs	Cluster of Workstations	5-3
NOWs	Networks of Workstations	5-3
CUS	Component Under Study	5-4
SS	Secondary Structure	5-6
H Test	Kruskal-Wallis H Test	5-15
df	degrees of freedom	5-15
CASP	Critical Assessment of techniques for Protein Structure Prediction	7-2
PHGA	parallel hybrid GA	H-2

Abstract

Interest in discovering a methodology for solving the Protein Structure Prediction (PSP) problem extends into many fields of study including biochemistry, medicine, biology, and numerous engineering and science disciplines. Experimental approaches, such as, x-ray crystallographic studies or solution Nuclear Magnetic Resonance (NMR) Spectroscopy, to mathematical modelling, such as minimum energy models are used to solve this problem. Recently, Evolutionary Algorithm studies at the Air Force Institute of Technology (AFIT) include the following: Simple GA, messy GA (mga), fast messy GA (fmGA), and Linkage Learning GA (LLGA), as approaches for potential protein energy minimization. Prepackaged software like GENOCOP, GENESIS, and mGA are in use to facilitate experimentation of these techniques. In addition to this software, a parallelized version of the fmGA, the so-called parallel fast messy GA (pfmGA), is found to be “good” at finding semi-optimal answers in a reasonable time. The aim of this work is to apply a (Multiobjective MO) approach to solving this problem using a modified fast messy GA. By dividing the CHARMM energy model into separate objectives, it should be possible to find structural configurations of a protein that yield lower energy values and ultimately more correct conformations.

In addition to the MO approach using the Multiobjective fast messy Genetic Algorithm (MOfmGA), various experiments are analyzed for effectiveness: newly designed Ramachandran plots, varied Building Block (BB) cutoff sizes and multiple competitive templates for both the fmGA and MOfmGA. Finally, an analysis of the efficiency using the pfmGA constructed with a farming model is studied. As these variants are expected to yield better results, so too is the first time implementation of the per residue Ramachandran plots. Following the analysis of these experiments, a comparison of previous methods is accomplished.

A MULTIOBJECTIVE APPROACH APPLIED TO THE PROTEIN STRUCTURE PREDICTION PROBLEM

I. Introduction

The Protein Structure Prediction problem is a Grand Challenge problem [14, 61]. Solving this problem involves finding a methodology that can consistently and correctly determine the geometrical conformation of any fully folded protein without regard to the folding process. The problem is simply stated; however, one must study the entire complexity of the problem to admire this *Gordian knot*¹.

The motivation for having the ability to find the conformation of a fully folded protein is in its application – a protein’s conformation represents a protein’s function [94]. Upon determination of the function of a particular protein, researchers may be able to engineer proteins for the making of particular products. Without the knowledge of the function of these proteins, these products would be impossible to construct. Interest in both the commercial and military realm is high in this area for the production of these engineered products. The military can use specially hardened material for body armor and plane shielding. Within the commercial world computer companies are constantly looking for material that allows computers to store more data and transfer communication signals faster. In addition to being able to engineer better materials, this research also supports the Human Genome Project (HGP). The HGP is supported by the National Human Genome Research Institute (NHGRI) at the National Institute of Health (NIH) and Department of Energy (DOE) [82]. This project’s goal is to identify the gene sequences for human DNA and to store this

¹A knot tied by Gordius, king of Phrygia, held to be capable of being untied only by the future ruler of Asia, and cut by Alexander the Great with his sword [25]

information within a database for later analysis. Moreover, the DNA is used for the process of building protein – once the structure of a protein is found, functionality may be mapped back to DNA-gene holding patterns making it easier to find possible weak or disease prone gene sequences within the DNA [86].

This Thesis effort studies the effectiveness and efficiency of a MOfmGA and fmGA when used to solve the PSP problem. Consequently, this analysis evaluates the conformations of two proteins: *[Met] – Enkephalin* (MET) and *Polyalanine*₁₄ (POLY). See Appendix N for atom and amino acid identification for each of these proteins. This introductory chapter discusses an overview of the problem, research goals, assumptions, risks, sponsorship areas and the thesis layout.

1.1 Overview of PSP Problem

Because the PSP problem is a biochemistry problem mapped to a computer for solving, it is necessary to describe the problem in both the biochemistry and computer science domains. In addition, constraints must be drawn within these domains to decompose the problem into a solvable entity. Following is a description of the problem in both domains.

A phenotype is the physical representation of the genetic code (genotype) [3]. Problems in the real world (Phenotype) are almost never readily encoded into computer terms. In fact, most problems have several levels of encoding before being delivered into an algorithm for evaluation. For example, the PSP problem encoding has different levels of encoding where the Phenotype representation is the actual physical structure of the protein.

Solving the PSP problem involves finding a methodology that can consistently and correctly determine the geometrical conformation of any fully folded protein without regard to the folding process. We must now consider the folding process. Proteins are constructed of a linear chain of amino acids. These amino acids are the building blocks or the foundation of every known protein. Every amino acid has a standard sequence of three atoms: one Nitrogen and two Carbons which make up the backbone of a protein: Nitrogen (N) , Carbon (C), and alpha-Carbon (C_α). In addition to these standard atoms and bonded to the alpha-carbon, is a residue (R) or side chain and a single hydrogen (H) atom. Discussed next is the generation process of a protein. It describes how the linear chain of amino acids is selected and joined together to form the protein. As the protein is created, the folding process begins.

Illustrated in Figure 1.1 is the process of protein generation. The picture depicts the linear sequence of amino acids being generated by a ribosome. The entire process beings within a cell when a

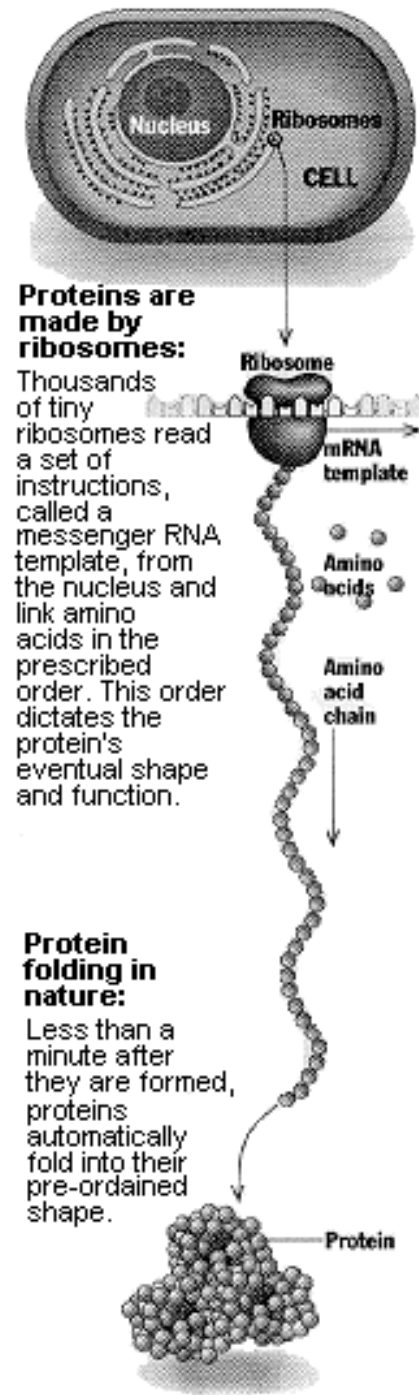


Figure 1.1 Description of the protein creation and folding process [15]

copy² of the DNA³ is made onto a deputy molecule called the messenger RNA⁴ (mRNA). This process of duplicating DNA is called *transcription*. Transcription is accomplished because the DNA carries the code for assembling amino acids into proteins. Once the mRNA has the properly transcribed sequence of Codons, the mRNA attaches itself to the ribosome and the translation process begins. *Translation* is the process where the production of the protein occurs. The ribosome travels along the Codons of the mRNA. As the ribosome encounters each Codon, a transfer RNA (tRNA) carries that particular amino acid that matches the encoded Codon to the ribosome.

This process aligns the amino acids in the correct order allowing for the amino acids to attach to one another making one long linear link of amino acids – and ultimately building a single protein⁵.

²A complete copy of the DNA is not made; moreover, only the information contained in the sequence of bases in the sense strand of DNA is impressed upon the mRNA

³DNA stands for deoxyribonucleic acid

⁴RNA stands for ribonucleic acid

⁵ $DNA \rightarrow RNA \rightarrow protein$

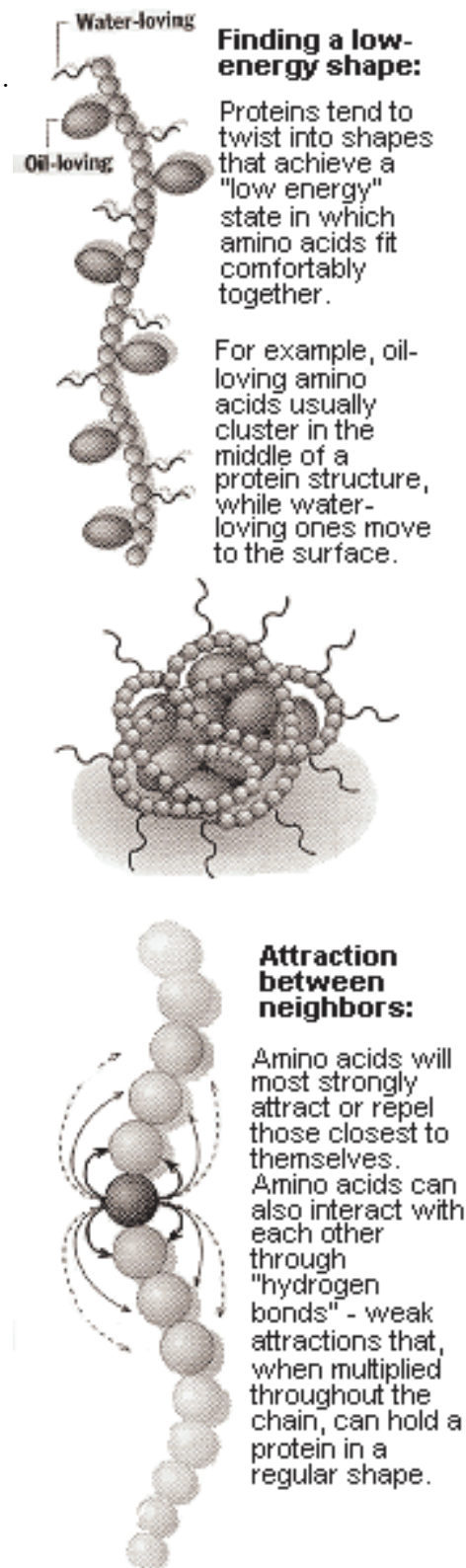


Figure 1.2 A few principles of protein folding [15]

As the protein is being generated, one amino acid at a time, it is also folding to a particular conformation. There are virtually thousands of different shapes to which a protein may conform [15]. Each conformation has a corresponding function, as explained earlier.

There are some principles proteins follow when folding. Protein mostly form into a conformation that acquires the lowest quantum mechanical energy or, in this case, potential energy (See Figure 1.2). In this shape, a protein is in its most stable condition; however, this is not to say that there are not proteins that fold to a higher energy for functional reasons. On a higher level, amino acids can attract or repel vicinity amino acids as necessary to reach this lower energy state; furthermore, on a low lever, bonded atoms making up each amino acid do the same. Additionally, some amino acids are attracted to water, hydrophilic⁶, while other's are attracted to oil, lipophilic⁷. This becomes important when the protein is folding because hydrophilic amino acids become outer amino acids protecting lipophilic amino acids inside the protein [44].

These native conformations are sought by researchers and, based on the physics of the problem, the protein itself; however, the protein finds its own native conformation. This research concentrates on the final folded conformation of a protein. It does not attempt to simulate the folding process where the amino acids and atoms twists, push, pull, and convolve into a final stable structure which is different computational problem.

1.2 *Research Goals*

The main goal of this research is to find improved methodologies for finding good conformations of fully folded proteins. A multiobjective approach has not been

⁶Hydrophilic amino acids are known as water loving; whereas, hydrophobic amino acids are water hating.

⁷Lipophilic amino acids are known as fat/oil loving; whereas, lipophobic amino acids are fat/oil hating.

applied to the PSP problem until now. In addition to the using this new approach, there are several secondary goals of this investigation. These secondary goals include experiments that observe variations to the fmGA. The first secondary goal is to investigate how the integrate the newly found Ramachandran plots for each residue type might effect the effectiveness of the algorithm. The second secondary goal is to observe how multiple competitive templates effects algorithm effectiveness. The third secondary goal is to study and validate increased efficiency on a farming model for the pfmGA. The last secondary goal is to integrate a RMS feature into the fmGA to give the researcher feedback on the RMS distance a found solution is from the accepted true solution. Although RMS has been accomplished by other techniques previously, this is the first time for integration into the algorithm software.

The approach taken for this research is the following:

Objective 1 Develop an improved understand of the PSP problem using new technology and supporting research.

1. Completely understand key problem domain concepts.
2. Known protein structure restrictions and boundaries.
3. Acknowledge previous methods of attack to solving the PSP problem to be sure not to overlap already accomplished research.

Objective 2 Develop a working knowledge of parallel programming concepts for application to the PSP problem domain and algorithm domain.

1. Know the difference between the types of parallel computer system architectures in the world and engage the ability to program on such systems. These practices include parallel programming libraries and to be able to identify the best library for a particular architecture.
2. Apply a new parallelization to the fmGA solving the PSP problem.

Objective 3 Develop a working knowledge of Evolutionary Algorithms (EAs), Evolutionary Strategies (ESs), GAs, Genetic Programming (GP), Evolutionary Programming (EP), Simulated Annealing (SA), and MO approaches to problem domains.

1. Apply a GAs to the PSP problem; furthermore, fully understand the fmGA and all its components.
2. Identify applications for EAs, ESs, GPs, SAs and EPs. Be able to relate terms between each algorithm type and know good applications for each.

Objective 4 Apply biochemistry constraints and restrictions to the search space for solving the PSP problem.

1. Ramachandran Plots have been applied to the PSP problem in previous work. However, until now, the application has been applied generically to dihedral angles for every type of residue. New technology has allowed for a more precise method of *assigning* angular areas to dihedral angles according to the residue type. These new Ramachandran Plots are referred to as Plots per residue type.
2. In addition to the constraints of the Ramachandran Plots mentioned above, there are angular values that may be assigned exclusively to dihedral angles on known folding habits.

These constraints can be referred to as bounding box filters on angular values making up the transformed *dihedral angular* conformation of a protein.

Objective 5 Use visualization techniques to depict the sparseness of good solutions in the fitness landscape.

1. Provide a new fitness visualization technique to facilitate the sparseness of good fitness values and good conformation within the *dihedral angle* search area.
2. Provide a fitness visualization technique to help validate the fmGA search technique. The plot should consist of an RMS and fitness axis. The plot should reveal that as the fitness drops, the RMS difference should move toward zero. If it does not, the model is not good.

Objective 6 Statistical analysis methods are used for determining the merit of solutions found by an algorithm.

1. Statistical analysis of answers need to be applied for the validation that found solutions are better or worse than previously found solutions.
2. The Kruskal-Wallis H test is used for the Analysis of Variance (ANOVA).

Objective 7 Finally, visualization techniques are used to present found conformations in a 3D graphical display of the protein.

1. Using a standard protein visualizer like VMD or RasMOL, the best solutions found should be able to be viewed.
2. Visualization technique requires a file generation mechanism built in to the running algorithm.
3. An explanation of a computational steering technique to help biochemist guide the search algorithm with user input.

1.3 Assumptions

As with most research, there are assumptions⁸ needing to be stated upfront. Often times there are research efforts building on past achievements as it is in this

⁸These assumptions listed here do not include all the constraints and assumptions needed to encode the proteins into solvable problems. These biochemistry assumptions and constraints are covered in Chapter 2.

case. Much effort went in to ensuring the fmGA, pfmGA and MOfmGA were working properly during this research. The fitness function code written by a previous student is also assumed correct. This function is based on the CHARMM version 22 software [11] ; however, it was ported from FORTRAN code to C in the early 90s. In addition to the fitness function being generated by previous students, so too were the parameter files that are used to define MET and POLY for the evaluation of energy by the CHARMM software. Finally, the assumption that the readers of this thesis are of sufficient background in the computer science, biochemistry, high performance computing (HPC), parallel computer architecture, GAs, and scientific experimentation to understand the discussion.

Generally there are risks involved when making these assumptions. After a problem is mapped to another domain, it changes and validity of the solutions change. There is a risk that after mapping this unusually complex problem into a computer solvable problem, the goodness of answers we could ever hope to find might be limited by our mapping and constraints.

1.4 Sponsorship

This thesis is sponsored by the Materials Directorate, Air Force Research Laboratory (AFRL) Wright Patterson Air Force Base (WPAFB), OH. This research lab specializes in discovering new techniques for building specially engineered materials. Material sought by the military normally focus on hardened type materials; however, other research is as important. The Department of Defense (DOD] has a special interest in optical limiting materials. These types of materials are in high demand for protection against lasers targeting pilots' eyes while they steer aircraft. Specially engineered polymer dispersed liquid crystals that can act as a switchable light shutter without disintegrating after laser contact is but only one type of material sought by this highly technological group. Furthermore, the PSP problem stems from this desire of biochemists to engineer materials such as the one mentioned above. A

biochemist having the ability to know the function of a protein beforehand, can produce these specially engineered products faster and more precisely than any other biochemist in the world. Finally, finding the solution could propel the United States far beyond the competitors within this market.

1.5 Thesis Layout

This Thesis is organized in the following manner. *Chapter 1*, this chapter, provides an overview of the PSP problem. Additionally, the objectives of the Thesis investigation and approach are presented. *Chapter 2* covers the background of past PSP research including previous work here at AFIT and elsewhere. The detailed PSP problem formulation can be found within this Chapter as well as the statistical analysis method used in the results and analysis of *Chapter 6*. *Chapters 3 and 4* describe the High to Low level design of the PSP problem being mapped to the algorithm domain and then into code. Following the design chapters is *Chapter 5* where the design of experiments is described. This chapter includes the justification for various experiments and selected statistical analysis method. Additionally, this chapter describes the process, number of tests, and presentation techniques. *Chapter 6* discusses the results, analysis, and compare them to previous research. Conclusions of this investigation can be found in *Chapter 7*.

II. Problem Domain Models

The PSP problem is a Nondeterministic Polynomial-Time (NP) complete problem. There is no known deterministic polynomial-time complexity algorithm available to solve it [85]. For this reason, computer engineers use stochastic algorithms which find semi-optimal solutions to the class of NP complete problems such as this one. GAs are one such class of stochastic algorithm. In fact, much has been written on solving the PSP problem using stochastic algorithms: [10, 70, 34, 33, 71, 72, 51, 52, 53, 54, 55, 69, 21, 76, 77, 18, 17, 16, 19].

Statement of PSP Problem

\hookrightarrow “Without regard to the folding process, determine the final resting conformation of a fully folded protein.”

In generating conformations of fully folded proteins, bond angle bending and stretching have been accounted for by two mainstream methods: *fixed* and *variable* [39]. When assuming variable bond lengths and angles [36, 102], the problem domain model becomes more difficult. Furthermore, using the *variable* method this model is constrained to using Cartesian coordinates [2] for a formulation of a fit model. The bond lengths, bond angles and dihedral angles all become variables making the problem more difficult. As opposed to using the *variable* method, one can use the *fixed* method where bond lengths and angles are fixed. This method relaxes the number of variables to use only dihedral angles and the problem domain model becomes simpler. The testing within this thesis investigation all use the *fixed* model.

This chapter covers the details of the problem domain. It begins with a discussion of the how a problem is decomposed into smaller parts and then describes a symbolic and 0/1 formalized problem domain description for the PSP problem. Following is a Phenotype to Genotype discussion and then a simple problem versus the real-world problem instantiation. The simple problem to real world problem

instantiation establishes the PSP problem’s search space. Moreover, a structural point of view using a different mapping of the PSP problem is then covered. Next, data structure decomposition and then a cartesian coordinate domain formulation is discussed. This chapter concludes by a discussion of the energy fitness function, CHARMM, which is used in this investigation.

2.1 *Problem Domain*

The problem domain decomposition and description is a general term for the process of taking a large set of small connected regions (in this case a protein and the atoms describing a protein) and grouping them together into a smaller number of large zones (dihedral angles) [100]. All decompositions have to satisfy certain hard constraints, but typically we are actually looking for an optimal decomposition among the huge number of possible solutions. A simple way to quantify the quality of a solution is to assign a score to each decomposition based on the value of some fitness function (we use CHARMM energy model as the fitness function) . If we take the convention of assigning low scores to good solutions, it is apparent that domain decomposition can be viewed as a constrained function minimization problem – much like a low energy search landscape.

2.1.1 Symbolic/Formalized Problem Domain Description. Every computer representable problem can also be embodied with sets and sequences using set theory notation. The following describes the input, transitional, and output domains of the PSP problem. In addition, necessary operators are defined as well as the 0/1 formulation. The input and output domains are in protein data bank (PDB) file format - including every atom in the protein. This format is the standard representation of a protein’s 3D conformation. The transitional domain is a limited or a constrained “subset” of the input domain. The reason for identifying these constraints is to seclude the variable angles from the fixed angles found within each amino acid. Therefore, manipulation of these angles should be considerably easier

yet, when evaluating the overall energy of the protein we may include the entire set of atoms within the protein.

2.1.1.1 0/1 Formulation for the PSP problem. Let $f(P_i)$ be the fitness evaluation for a particular protein P_i . The fitness evaluation, Equation 2.1, calculates all t terms of a given energy function E_i using P_i represented by an input atom set $x(A)$. According to our CHARMM model, t is 8; however, this may change according to the energy function employed. Figure 2.8, starting with the third atom, every atom has a 2D matrix of bit-degrees running from 0^\bullet to 1024^\bullet and 0^\bullet to 512^\bullet . This is illustrated in Figure 2.1.

$$f(x) = E_0(x(A)) + E_1(x(A)) + \dots + E_t(x(A)) \quad (2.1)$$

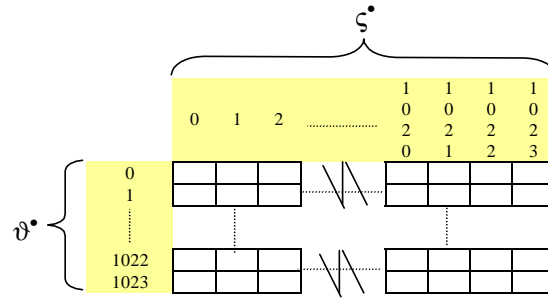


Figure 2.1 2D matrix representing the allowable angles for each set up 3 atoms defined by Figure 2.8.

Let $\Xi = [\xi_{ijk\dots n}]$ be a matrix of n dimensionality. Embedded within each subscript for ξ is the matrix shown in Figure 2.1. These matrices hold each and every discretized configuration possible for the atoms within the protein P_i . Allocation of values within matrices so that $\xi_{ijk\dots n}$ is 1 if angles ζ and ϑ are valid angles for the atoms i, j , and k ; otherwise $\xi_{ijk\dots n}$ is 0.

$$[\textit{!htb}]\xi_{ijk\dots n} = \begin{cases} 1 & : \zeta \text{ and } \vartheta \text{ are valid} \\ 0 & : \zeta \text{ or } \vartheta \text{ are invalid} \end{cases} \quad (2.2)$$

This is rather difficult to visualize because of the number of dimensions that are involved with even the smallest of proteins; however, patterns form within the matrices according to biologically invalid atom conformations. For example: in Figure 2.2 the lower atom is being bent back toward top atom, ϑ having the bit-degrees angle of one. It is known that atoms remain a certain distance apart from one another [94]; therefore, a configuration with ϑ having a value of one bit-degrees never occurs in the real world. The resulting conformation of protein i , $C(P_i)$, is infeasible and matrices are filled with zeros where these invalid angles are found. The objective would then to be minimize Equation 2.3.

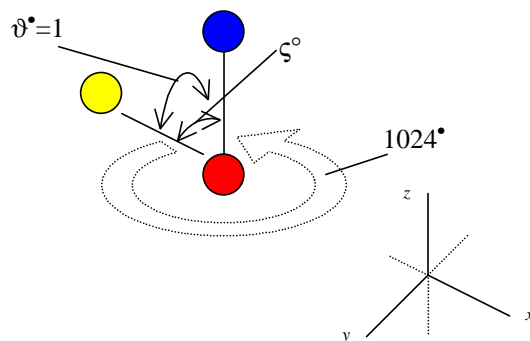


Figure 2.2 ϑ is shown to have the invalid bit-degrees angle of one. This supports the need to identify these angles as being unrepresentable within a conformation of a protein.

$$\forall_i \forall_j \dots \forall_n \sum_{h=1}^n \frac{f(X)}{\xi_{ijk\dots n}} \quad (2.3)$$

Equation 2.3 varies all angles for every combination possible for atoms within the protein. It is important to note that any set of angles not corresponding to the fixed angles within a set of atoms spread zeros throughout all matrices using that invalid angle sequence. Equation 2.3 ensures that illegal angles reflect a high energy value by forcing it to infinity. Computers may generate a divide by zero error; however, this is key in knowing we have an ill-formed protein.

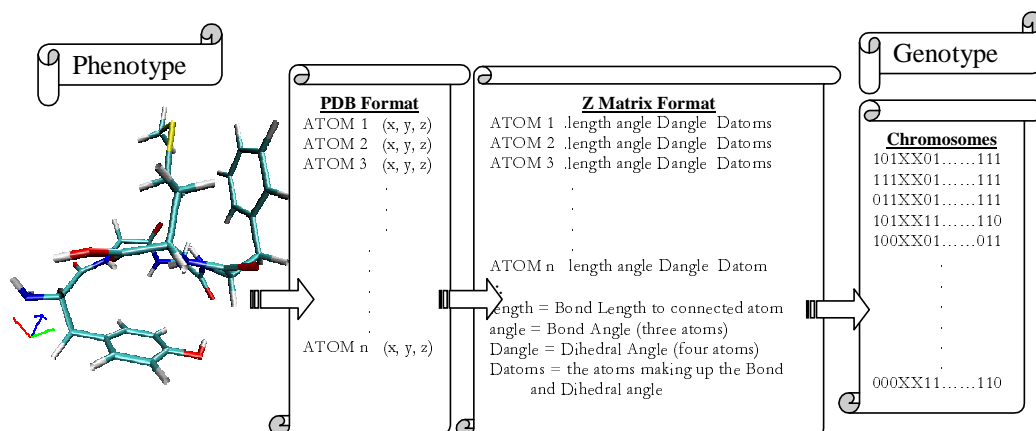


Figure 2.3 Illustration of Genotype to Phenotype domain

2.1.2 Genotype. Generally speaking, the genotype of an individual is that individual decomposed into parts or building blocks. This can be represented as different types of data blocks or data structures within a computer. For the most part, this coincides with how the problem is encoded and, in all cases, the makeup of the genotype (whether it is a binary number, real number, or object encoding) is driven by the algorithm domain used to solve the problem.

2.1.3 Phenotype. The phenotype is the physical representation of the genetic code in the algorithm domain (genotype). Problems in the real world (phenotype) are almost never readily encoded into computer terms. In fact, most problems have several levels of encoding before being fed into an algorithm for evaluation. For example, the PSP problem encoding has different levels of encoding where the phenotype representation is the actual physical structure of the protein. In Figure 2.3 the phenotype is illustrated on the left side with the picture of a protein and the genotype is illustrated on the right side as that protein is represented within the computer in bit string form. The mapping from phenotype to genotype domain is described with the arrows and formats between each domain.

2.2 Genotype and Phenotype of the PSP problem

The genotype and phenotype are defined above; however, many details have been left out of the PSP problem's genotype and phenotype. This section continues the discussion and defines the biochemistry domain (phenotype) and the offered algorithmic solution (genotype) as they relate to this thesis investigation.

2.2.1 Diagrams/Graphs for insight to problem domain. Proteins are constructed of a linear chain of amino acids. This is illustrated in Figure 2.4. Generally speaking each amino acid in a protein is linked into what can be called the protein's backbone structure [8] with the following sequence of backbone atoms: nitrogen (N), alpha-carbon, and carbon (C) (this is illustrated in Figure 2.5). Additionally, bonded to C_α are residue, R, or side chain and hydrogen (H) atoms. Furthermore, a protein is constructed with a sequence of these amino acids. In addition to knowing the makeup of amino acids, it is known they exist in 20 different configurations (See Appendix A and B) each with its own unique side chain (atom sequence), R [94]. One can picture a protein's backbone starting with N and running through the sequence of atoms within each amino acid until it ends with the last C in the last amino acid. The sequence for a four amino acid protein would look like: N- C_α -C-N- C_α -C-N- C_α -C-N- C_α -C. Each sequence of three atoms (N- C_α -C) has the structure shown in Figure 2.5 plus a side chain. A sequence of four amino acids is illustrated in Figure 2.6. The number of shapes that even a small protein can take on would be quite large. At this point, the sequence of atoms, amino acids, and bonds has been defined; furthermore, we even know from Chapter 1 how a protein is built. But, what we don't know is how this protein functions. Protein functionality is sought for many reasons. One such reason is that by finding a particular protein's function it may lead to curing a disease [49]. With proteins, conformation equals function [94]. Therefore, if we want a protein that has a specific function, we need to know its final conformation before generating it. This brings us back to the original problem, what is the final conformation of a fully folded protein?

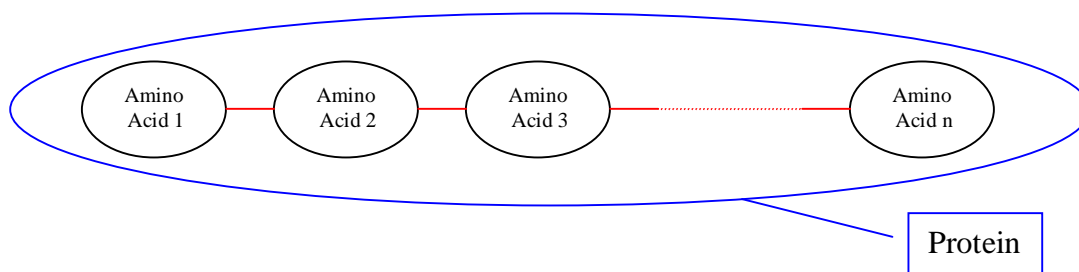


Figure 2.4 High Level abstraction of a protein consisting of n amino acids.

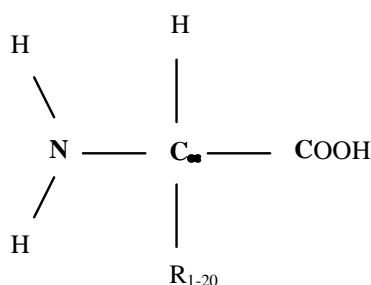


Figure 2.5 Un-ionized form of an amino acid

These complex structures are known to fold in a matter of seconds [15]; furthermore, the protein configuration can be complex that physical models of tiny proteins folding cannot be emulated on today's HPCs [67].

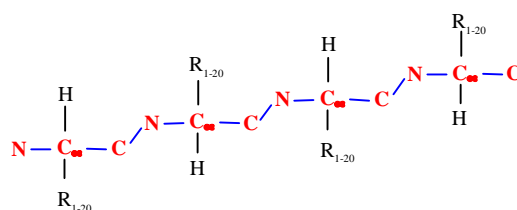


Figure 2.6 Representation of Protein revealing atoms bonded together to make the linear chain of Amino Acids

2.2.2 Simple problem vs. real-world problem instantiation. The following simple model is used to get an appreciation of the search space complexity or the number of different conformation a protein might take on. For this scaled down example, consider a protein with only three atoms like in Figure 2.7.

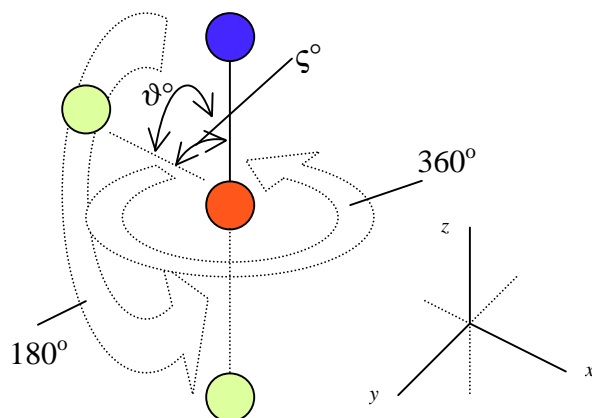


Figure 2.7 Simple three atom model

This illustration has circles representing atoms. The lower atom is shown to be able to rotate. It can rotate 180 degrees about the x-axis and 360 degrees about the z-axis. It is shown in Figure 2.7 that by limiting the angles to be integers values, the number of combinations to evaluate are $360^\circ * 180^\circ$ equaling 64800. Accordingly, if we were looking for the best configuration, we must check each of the 64800 different combinations. This may not seem to be difficult. In fact, a computer might be able to run through a small number of combinations like this in a matter of seconds depending on the fitness evaluation cost for each combination. Unfortunately, atom conformations are not restricted to having integer-valued angles - they can have both rational and irrational numbers as angle values. Consequently, atoms in a protein may have an uncountable amount of different angles. This drives the number of total possible combinations up to infinity. Clearly a computer cannot check the entire search space. Moreover, computers have a limit to the size of search space mainly because they are limited by the number that can be represented. At the hardware level, the number of bits in one register is limiting; moreover, at the software level, each language has its own variable type limitations. As a result, bounded by the limitations of a computer, we must devise a means to manage the input in a way that a computer might have a chance to solve such an intractable problem.

A choice between real and binary values is required. In the past both of these encodings yield similar results [68, 9, 32, 35, 50, 20, 74]. Thus a binary encoding is chosen and the angles are discretized into 1024 (1Mbyte or 2^{10}) sections for every 360° . In terms of our simple problem shown in Figure 2.7, we would have different combinations. In Figure 2.8 it is shown that the discretizing has given a new meaning to angles between atoms. Replacing the normal 360° for a circle, we have 1024^\bullet . Where $^\circ$ denotes degrees and $^\bullet$ denotes bit-degrees. It can be shown that although we have increased our number of combinations to evaluate, the problem has been transformed into a computer solvable problem.

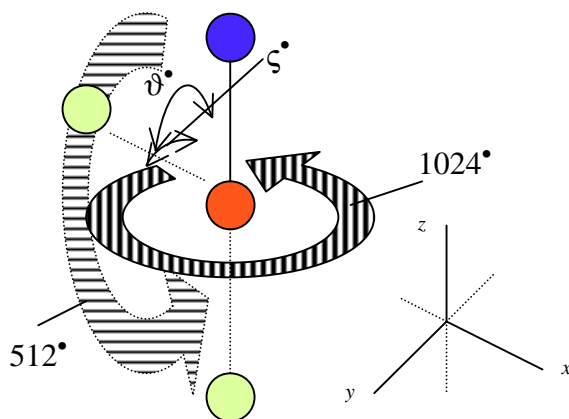


Figure 2.8 Simple three atom discretized model

Extending this simple example to that of a real protein, which normally is made of several hundred atoms, we can see how the number of combinations grows according to the number of atoms in a particular protein. A protein with three atoms has $2^{10} * 2^9$ or 2^{19} different combinations; furthermore, a protein with four atoms (see Figure 2.8) has $2^{19} * 2^{19}$ or 2^{38} different combinations in the search space. Generalizing, a protein with n atoms, where n is two or more, has $2^{19*(n-2)}$ combinations in the search space.

$$2^{19*(n-2)} \quad (2.4)$$

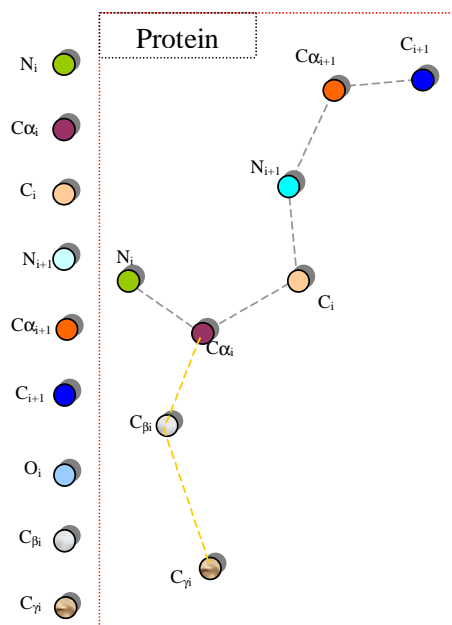


Figure 2.9 Fictitious protein

This is a large search space. A computer using a deterministic search to find the final folding state of a twelve-atom protein, worst case, would have to look at 2^{190} different structures before knowing that, within the limits of the computer, the best structure has been found.

Discussed earlier in Chapter 1 are principles of protein folding [15]. The first principle listed is the one that specifies that the 'lowest energy' level also indicate the correct geographic conformation. In nature, proteins fold to this shape automatically and within seconds. Unfortunately, predicting this shape has been extremely difficult because the number of variables involved increases exponentially according to the number of atoms found in the protein. As you can see in the 'Attraction between neighbors' image in Figure 1.2, each amino acid interacts with every other amino acid. These are just a few of the physical rules biochemists have discovered about proteins. It is said that these findings combined together make up the chemistry of the PSP problem.

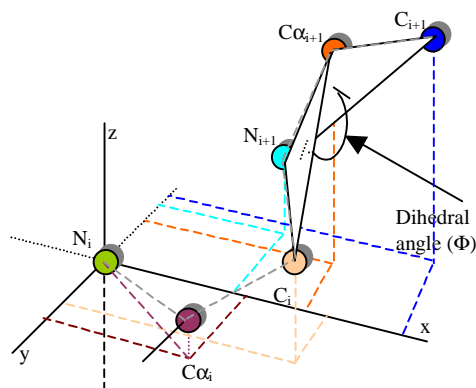


Figure 2.10 ϕ angle representation.

2.2.3 A Structural view point of the PSP problem. Following the decision to use a fixed bond angle and length method we are left with the dihedral angle as the only variant. A brief discussion of how they are developed is required. Furthermore, formulation to encode the PSP problem must also include a dihedral angle encoding to maintain a fit model. A dihedral angle can be found given any set of any four bonded atoms – each with x, y, and z coordinates (Figure 2.3 illustrates and Equations 3.9-3.17 describe the mapping process.). When given a protein’s 2-dimension chemical formula or amino acid sequence we are also being handed the amino acid sequence for that protein. It is this amino acid sequence that also fully describes the entire atom layout. So, now given just the amino acid sequence of a protein we should be able to encode the protein in a dihedral angle sequences that, after a stochastic search, fully describe the structure of the protein. Going back to what we already know about amino acid sequences and the pattern of N- C_{α} -C atoms from each amino acid making up the backbone, we can further define a set of three dihedral angles, (ϕ , ψ , and ω), one for pattern of four atoms going down the backbone of the protein. For example: If we were given a protein, such as the one in Figure 2.9, and we wanted to encode it into dihedral angles using bit degrees, we find angles associated with every sequence for four atoms running down the backbone. The ϕ angle would be found using atoms N_i , C_{α_i} , C_i , and N_{i+1} , which is illustrated in Figure 2.10.

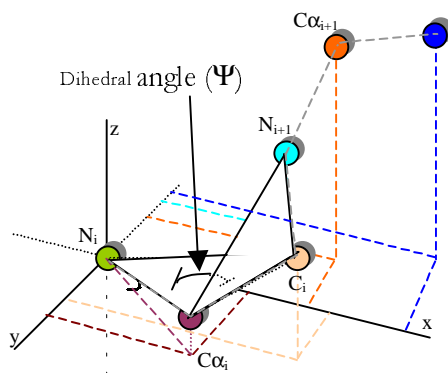


Figure 2.11 ψ angle representation

The ψ angle would be found using atoms $C_{\alpha(i)}$, C_i , N_{i+1} , and $C_{\alpha(i+1)}$ which is illustrated in Figure 2.11. Lastly, The ω angle would be found using atoms C_i , N_{i+1} , $C_{\alpha(i+1)}$, and C_{i+1} which is illustrated in Figure 2.12. Now that the backbone is specified into angles, they can each be converted into 1024 bit degrees. The second hurdle is encoding each side chain or residue hanging off the backbone. We do this in the same manner as we did the backbone. Starting with the three, already specified, backbone atoms plus the first atom making the side chain, each atom of the side chain is then specified one at a time using three previously specified atoms until each atom is specified with unique dihedral angle. Considering there is 20 different side chains, each offering a different number of dihedral angles, these angles are labelled $\chi_1, \chi_2, \dots, \chi_n$ respectively. Figure 2.13 illustrates the $\chi(2)$ dihedral of a make believe side chain hanging off of the protein we have been using in our example.

2.2.3.1 Data structure Decomposition. The real world data decomposition discussed is an encoding of the problem into a computer solvable problem; however, there are more steps to map the problem from the dihedral angle into a workable string of bits for the fmGA to use in the search for the lowest possible fitness value. Equation 2.5 is this transfer function.

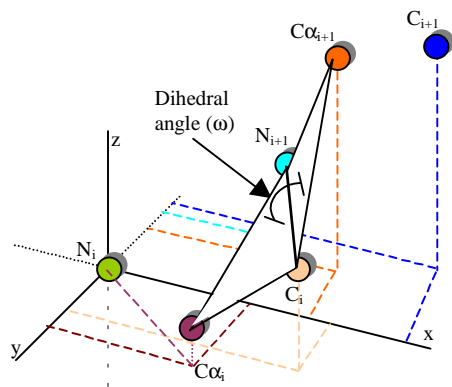


Figure 2.12 ω angle representation

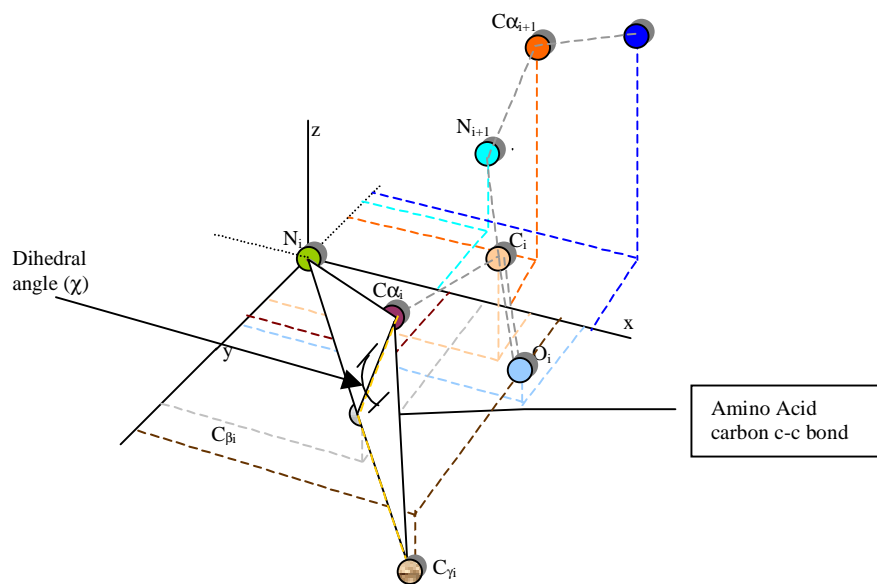


Figure 2.13 χ angle representation

$$AminoAcids_1 \rightarrow Atoms_2 \rightarrow Dihedrals_3 \rightarrow BitStringRepresentation_4 \quad (2.5)$$

Equation 2.5 describes the data structure decomposition down to the bit representation. During the algorithm search, the bit representation is changed frequently and after each change a evaluation of merit is needed. This need for an evaluation prompts a second transfer function needed to recompose the data in a manner which can be evaluated for merit. The following function resolves the atoms into two different representations and describes this type of transfer:

$$[x, y, z]_1 \Rightarrow [a, b, d]_2 \Rightarrow [\phi, \psi, \omega, \chi_i]_3 \Rightarrow [0, 1]_4 \quad (2.6)$$

Equations 2.5 and 2.6 are transfer functions describing data decomposition and format decomposition on a high abstract level. The evaluation of fitness occurs when the atoms are described by their Cartesian coordinates (Indicated by Subscript 1 of Equation 2.6).

2.3 Domain Formalization for cartesian coordinates

- Input Domain

$P_i \equiv$ Input protein where protein $P(A, B)$, where A is the set of atoms and B is the set bonds connecting atoms. A is the x-tuple having the x-y-z coordinates and properties of that particular atom. B is a y-tuple having properties identifying bond types. $M_i \equiv$ Input set of known amino acids patterns $M(A^2)$ where is a known amino acid formula.

- Transitional Domain

$\sim P_j \equiv$ Processed protein where $P(A^2)$ is protein P_i after it has been broken up into the set of amino acid sequences and then into the set of atom sequences,

A^2 where A_i^2 is a 3-tuple holding the x-y-z coordinates of atom i, A_i^2 , with respect to (WRT) the previous atom. Where $A_0^2 = (0,0,0)$ because it is the atom at the origin.

- Output Domain

$O(A^2) \equiv$ Final atom coordinates. A_i^2 is a 3-tuple holding the x-y-z coordinates of amino acid i, A_i^2 , WRT the previous atom.

- Operators

$D(X) \equiv$ Dihedral angle calculation given a set of four atoms.

- Feasible/Optimal Conditions

$E_j(P_i) \equiv$ Energy calculation for a particular protein conformation. $f(X)$, input conditions where the sum is the fitness of a solution $f(x) = \sum_{j=1}^t E_j(P_i)$. $C(P_i)$ Feasibility check for valid angles between all atoms in protein P_i .

As described in Section 2.2.3, Dihedral angles are described using four atoms.

Figure 2.14 depicts four atoms and their corresponding dihedral angle.

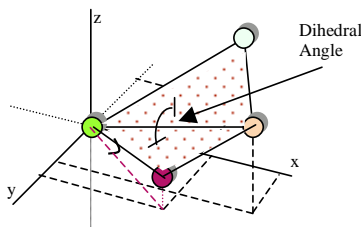


Figure 2.14 Simple example of four atoms making a measurable dihedral angle.

- Energy Calculation

Figure 2.15 illustrates numerous positions bonded atoms might have in a protein. In addition, it plots how these configurations influence the potential energy calculation. The balls are representing atoms and graphed curved lines (on the right of each position) are identifying the interacting variable affecting the potential energy for that conformation. Each of these conformations

occurs between each bonded atom within a protein. Each of these six functions makeup the energy function used to calculate the *fitness* of a particular conformation.

Empirical Potential Energy Function

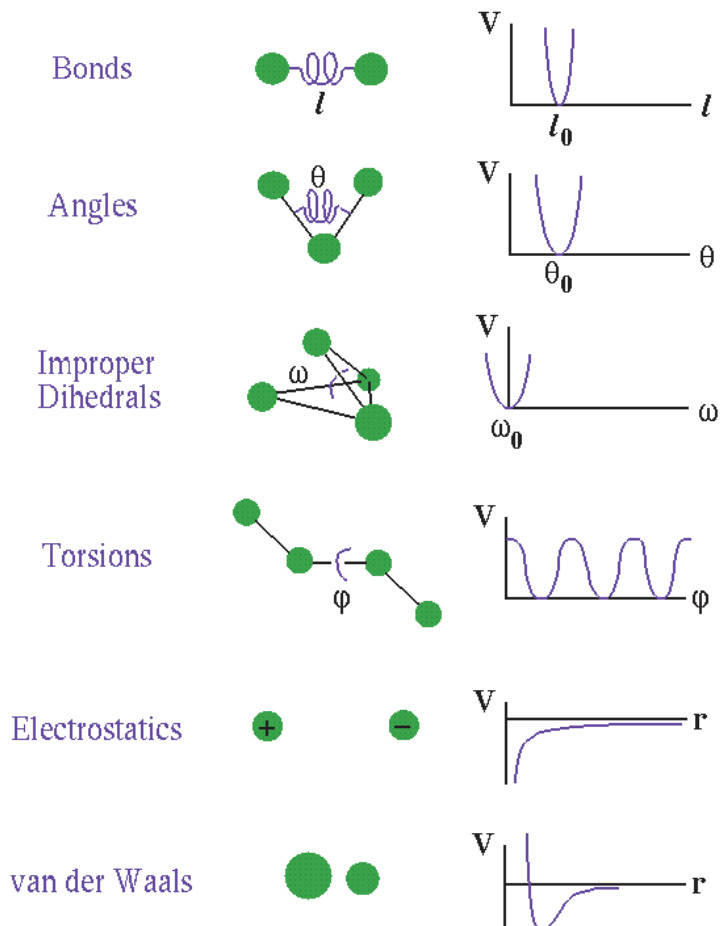


Figure 2.15 Graphical description of energy functions and how they are translated from physical atom-bond relationships to Potential Energy Functions.

2.4 Importance of Energy Function

Search algorithm rely solely upon the ability to be able recognize good solutions. For the PSP problem, this recognition comes in the form of an energy function or fitness functions. Solutions found to be dominate by one may be found to be weaker by others using a different fitness function. This is why it is extremely

important to have the most suitable fitness function for the problem. This suitability is particularly difficult to achieve for the PSP problem. Many factors are involved in choosing an suitable energy function. Potential energy [96], quantum mechanical energy, chemistry of the protein [89] [37] [38] [5] [99], empirical force fields energy, energy surface with the largest statistical weight [37] and entropy [80] are just a few of the fitness function ingredients that may be used. This thesis effort uses the CHARMM Energy model. CHARMM is a potential energy model where minimization search techniques are employed.

2.5 CHARMM ENERGY FUNCTION

Essentially the CHARMM energy function sums the internal terms or bonded atom energy and external terms or non-bonded atom energy of a particular protein in a specific conformation.

$$E_{total} = \sum_{(i,j)(connect)} E(bonded) + \sum_{(i,j,k,...,n)(!connect)} E(non - bonded) \quad (2.7)$$

Bonded energy is the sum of bond stretching, bond rotation, bond bending, improper torsion and hydrogen bonding energy reduction between each connected or bonded atom.

$$E_{stretching} = \sum_{(i,i+1)} K_b(b - b_0)^2 \quad (2.8)$$

Where K_b is the force constant determining the strength of the bond, b is the actual bond length and b_0 is the ideal bond length (Equation 2.8). The bending energy is similar to that of the stretching energy where K_θ is the force constant, θ is the actual measured angle, and θ_0 is the ideal angle. This is primarily a penalty function (Equation 2.9).

$$E_{bending} = \sum_{angles(i,j,k)} K_\theta(\theta - \theta_0)^2 \quad (2.9)$$

The third term in the bonded energy calculation representing the accounts for a reduction in the van der Waals term between the hydrogen atom and the acceptor atom [11] (Equation 2.10).

$$E_{hydrogen} = \sum_i \left(\frac{A'}{r_{AD}^i} - \frac{B'}{r_{AD}^i} \right) \cos^m(\theta_{A-H-D}) * \cos^n(\theta_{AA-A-H}) \quad (2.10)$$

The fourth term in the bonded energy calculation representing the torsion angle potential function which models the presence of steric barriers between atoms separated by 3 covalent bonds (1,4 pairs) is shown in Equation 2.11.

$$E_{torsion} = \sum_{(i,j,k) \in D} K_\theta (1 - \cos(n\phi)) \quad (2.11)$$

$$E_{Improper-torsion} = \sum_\omega K_\omega (\omega - \omega_0)^2 \quad (2.12)$$

Equations 2.8, 2.9, 2.10, 2.12 and 2.11 make up the energy for bonded atoms:

$$E_{bonded} = E_{torsion} + E_{bending} + E_{hydrogen} + E_{stretching} + E_{Improper-torsion} \quad (2.13)$$

The final terms for the calculation of energy are the non-bonded related terms, electrostatics, water-water interaction and van-der-Waals. These terms may be combined into the following sum:

$$E_{lennard-jones} = \sum_{(i,j) \in \mathcal{N}} \left[\left(\frac{A_{ij}}{r_{ij}} \right)^{12} - \left(\frac{B_{ij}}{r_{ij}} \right)^6 \right] \quad (2.14)$$

Constants A and C are interaction energy using atom-type properties. D is the effective dielectric function for the medium and r is the distance between two atoms having charges q_i and q_k .

$$E_{electrostatics} = \sum_{(i,j) \in \mathcal{N}} \left[\frac{q_i q_j}{D r_{ij}} \right] \quad (2.15)$$

$$E_{water-water1} = \sum_i K_i (r_i - r_{i0})^2 \quad (2.16)$$

$$E_{water-water2} = \sum_i K_i (\theta_i - \theta_{i0})^2 \quad (2.17)$$

Contributions of water-water constraints of distance and dihedral angles are shown in Equation 2.16 and 2.17 respectively. Furthermore, the entire contribution of non-bonded energy is given by equation 2.18.

$$E_{non-bonded} = E_{lennard-jones} + E_{electrostatics} + E_{water-water1} + E_{water-water2} \quad (2.18)$$

The CHARMM energy function is quite computationally expensive. In Table 2.1 a comparison of CHARMM, AFIT CHARMM, Amber, ECEPP, and Optimized Potentials Potentials for Liquid Simulations (OPLS) is illustrated. The coding details and objective decomposition is discussed later. Notice that CHARMM covers each one of the possible energy equations; however, AFIT's CHARMM has reduced this function due to the insignificance of these other forces. AFIT CHARMM was used in this Thesis investigation and has been found to be a valid model in the past [65].

In addition to these energy models many other models have been used for other approaches. The Random Energy Model (REM) was applied to the PSP problem by Bryngelson and Wolynes [12]. This energy model was originally used in spin glass theory [22]. Other such fitness function models have been applied to the PSP problem using enthalpy [80], conformational entropy, hydrophobic/hydrophilic [84], and distance matrix models employing Frobenius norm of differences, Hoeffding inequality keeping corrected distances for fitness function terms [84], and ring closure on local conformations [38]. Moreover, all these models have the same theme in trying to define the properties a real protein has when folding. Today, it seems that no single model has prevailed and the search for the perfect fitness model continues.

	Equation # \rightarrow	2.8	2.9	2.11	2.12	2.14	2.15	2.10	2.16	2.17
Acronym	Name									
CHARMm	Chemistry at Harvard using Molecular Mechanics	X	X	X		X	X	X	X	X
AFIT CHARMm		X	X	X	X	X	X			
Amber	Assisted Model Building with Energy Refinement	X	X	X		X	X	X		
ECEPP/3	Empirical Conformational Energy Program for Peptides			X		X	X	X		
OPLS	Optimized Potentials Potentials for Liquid Simulations			X		X	X			

Table 2.1 Comparison Of Common Energy Functions used in solving the PSP problem [96] [87] [20]

2.6 Summary

This Chapter has discussed the PSP problem domain. Using a phenotype to genotype mapping, it has delineated the problem in computer terms and constraints. Furthermore, it established that this is a minimization problem (which every problem can be converted to such a problem [73]) and the fitness function to minimize is defined. Chapter 3 covers different types of tools to solve the PSP problem and then maps this problem.

III. Possible Algorithm Domains

“To the man who only has a hammer in the toolkit, every problem looks like a nail.”

-Abraham Maslow

The universe is in a continual state of change – evolving such that the entire process, life, is in flux. Evolution defines this universe with levels of abstraction called *selection* or chance of survival. When attempting to capture these levels of the evolutionary process, the researcher mimics, in computer language, each level of this process as they are understood. These levels of abstractions *personify* parts of the evolutionary process; furthermore, each computation stratagem discussed contain a subset of these levels of abstractions. This Chapter is a discussion on methods used for characterization of the evolutionary process. All types of methods are covered direct measurement methods to full simulations of the folding process. Following this is a short discussion of methodologies used at AFIT and a problem domain to algorithm domain mapping.

3.1 Background

Approaches to finding the structure of a fully folded protein are numerous. They range from software to hardware driven, theoretical to empirical, and fine to coarse-grained. To highlight the more generic methods, the following are discussed: X-ray crystallography [94][88], molecular dynamics, nuclear magnetic resonance spectroscopy [48], Monte Carlo analysis [83], atomistic and non-atomistic lattice simulation, off-lattice simulation 3.6 and genetic or evolutionary algorithm approaches. These methodologies are classified in Table 3.1. There are many reasons for having a variety of approaches. Some protein conformations are easily found using empirical methods, like x-ray crystallography, because they crystallize easily, yet others are found in solution using nuclear magnetic resonance spectroscopy [94]. The time involved in finding structures using these empirical measurement techniques may be sometimes cost prohibited. In addition to the time investments, all approaches have

resolution limitations making the decision maker choose between measurement precision and time investments. These are just a few of the reasons there is a requirement for having alternatives methods when identifying the protein resting. Also, these approaches are often used to compliment one another. The following is a discussion on these various approaches.

	Experimental Methods	Simulation	Energy Minimization
X-Ray Crystallography	X		
NMR Spectroscopy	X		
Molecular Dynamics		X	
Monte Carlo Analysis		X	
Lattice Simulations		X	
Off-Lattice Simulation		X	
Evolutionary Algorithm			X

Table 3.1 Classification of methodologies used in solving the PSP problem. For advantages and disadvantages see Sections 3.2, 3.3, and 3.4.

3.2 Experimental Methods

Experimental methods require specialized equipment to measure the physical three dimensional structure of the protein. In addition to expensive equipment, these techniques also require many hours from expert technicians to discover protein conformations.

3.2.1 X-Ray Crystallography. X-ray crystallography is an empirical approach that is composed of three components: source of x-rays, a protein crystal, and a detector or x-ray film. An illustration of equipment used for x-ray crystallography at Lawrence Livermore National Laboratory (LLNL) is shown in Figure 3.1. To generate highly ordered crystals one could use slow salting of protein in a solution [94][88]. It should be noted that crystallization of proteins is an art and can take large amounts of time and patience; furthermore, some proteins cannot be crystallized¹. Once a crystal is obtained, it is then mounted between an x-ray

¹Approximately 65% that crystallize rarely or never [88]

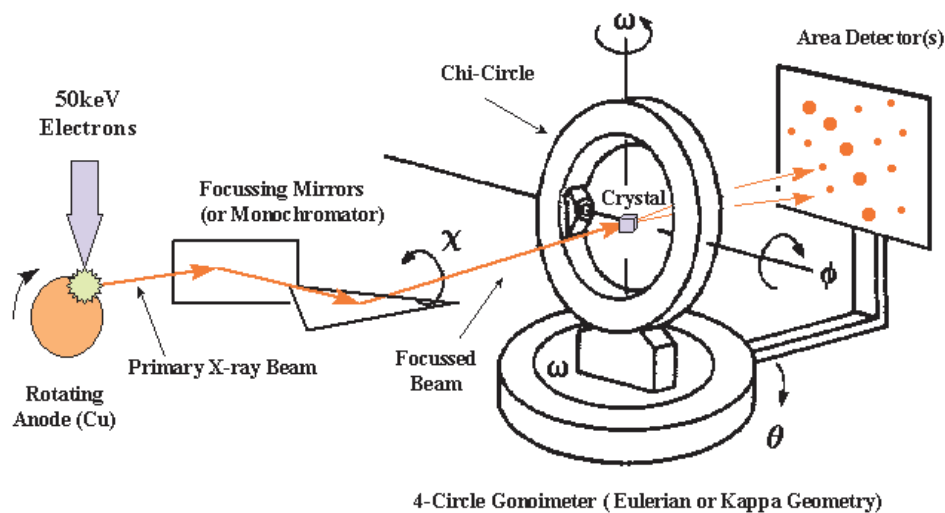


Figure 3.1 X-ray Diffraction Equipment found at Lawrence Livermore National Laboratory [88]

source and detector. The narrow x-ray beam is then passed through the crystal - scattering occurs [88]. The detector then records this scatter like film records a picture. Finally, after data has been collected the detectors pattern is described by a applying a mathematical relation called a Fourier transform. The output comes in the form of an electron-density map (contoured plot). The map can then regenerate the placement of each atom within the protein. This method is extremely effective and has been the major source of protein structure identification to date. It is known that it takes skilled scientists 2 to 3 months to have a protein's structure if given a crystal of that protein [27]. Currently, X-ray crystallography describes over 80% of the protein structures deposited in the protein databank [26]. In Figure 3.2 it is illustrated that in 2001 there are 3298 structures added. This means that at least 2600 are found using X-ray crystallography. At this rate, researchers can find about 216 proteins a month [26].

3.2.2 Nuclear Magnetic Resonance Spectroscopy. Another method called NMR Spectroscopy can reveal the protein's conformation while the protein is in a solution. Normally, this method is used to in conjunction with x-ray crystallogra-

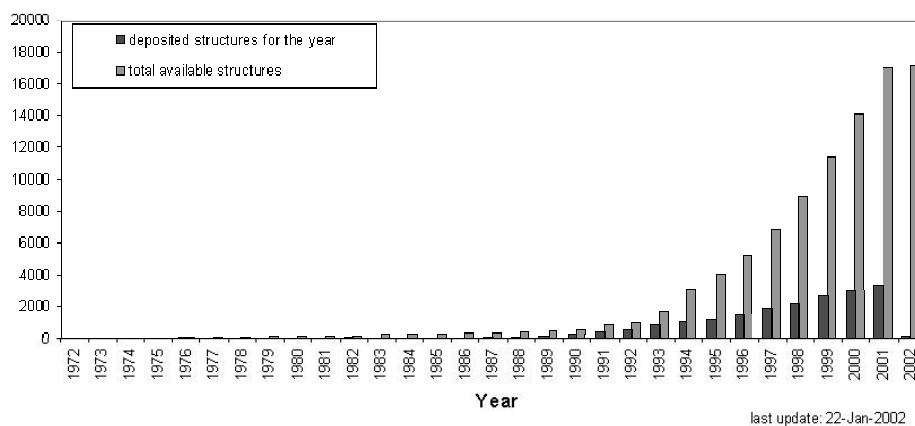


Figure 3.2 Year vs Total Available Structures held in the Protein Data Bank [26].

phy. This method alone only contributes to about 16% of the total amount deposited structures in the Protein Data Bank [26]. It is based on the knowledge that hydrogen's atomic nuclei are intrinsically magnetic and have a measurable energy state with the application of a magnetic field. Depending on the orientation of the hydrogen atom at the time of observation there are two states of observed hydrogen: alpha and beta [48]. Accordingly, these two states emit different energy levels - the difference between these states is proportional to the strength of the applied field. Using this technique, a multidimensional NMR spectroscopy approach can nail down orientations of hydrogen molecules along the backbone of a protein thereby allowing researchers to discover the structure of the evaluated protein. The advantage of this method over x-ray crystallography is that there is no need for a protein crystal - the conformation is simply found by placing the protein within solution. However simple, this method is limited by obtainable resolution and the time to get results. As compared to x-ray crystallography, x-ray crystallography can obtain higher resolution than NMR, but it takes longer time range to do so.

3.3 Simulation Methods

Simulation methods are computer programs that attempt to solve the PSP problem by simulating the folding process. These methods normally use extensive

computational resources and require considerably more time than energy minimization methods. Moreover, corporations like International Business Machines (IBM) Corporation has recognized the need for extensive computational resources to run these experiments and is gathering data for the constructing a petaflop computer specifically designed for simulating the protein folding process [23][64].

$$CPU_{speed} = 4000^n 10^{15} flops \quad (3.1)$$

In examination of IBM's petaflop computer chances of solving the PSP problem we turn to what is known about the computational requirements for a simulation. It is known that the time steps required to accurately account for thermal oscillations of the protein are on the order of one femtosecond (10^{-15} sec) [66:5–7][32]. Therefore, if a single calculation between two atoms must be computed within a femtosecond, the number of calculations required for a single pair combinatorially rises as the number of atoms is increased. For example, if it takes 4000 flops to calculate one quantum mechanical function between two atoms, the number of floating operations per second (flops) require would grow exponentially $4000^{number\ of\ atoms}$ per femtosecond for a real time simulation. Equation 3.1 illustrates formula for finding the a CPU's computational speed requirement, where n equals the number of atoms for a particular protein. IBM's petaflop computer has not been constructed yet and it still is not going to meet a protein simulation's computational need. To date, these methods where used in finding 2% of the total number of proteins found in the Protein Data Bank [26].

3.3.1 Atomistic and Non-Atomistic Lattice Simulations. It is necessary to define the components of lattice space. A space lattice is an infinite, three-dimensional periodic arrangement of mathematical points. Lattice space is a mathematical model having points that can represent an atom or group of atoms according to how the model is to represent a given model. In this case, the model is a protein

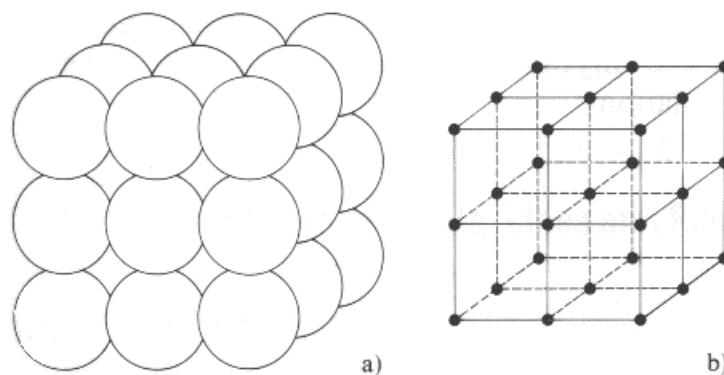


Figure 3.3 An example of using a lattice space model (*b*) to represent atoms in crystal form (*a*) [91].

having a set of definable atoms. A lattice space mathematical grid point could represent either an atom or an entire amino acid. Furthermore, a lattice model would be layered having sub lattice space models within each higher level model. In the scientific visualization world this concept is called world within worlds [81]. The following list explains the difference between Atomistic and Non-Atomistic Lattice models.

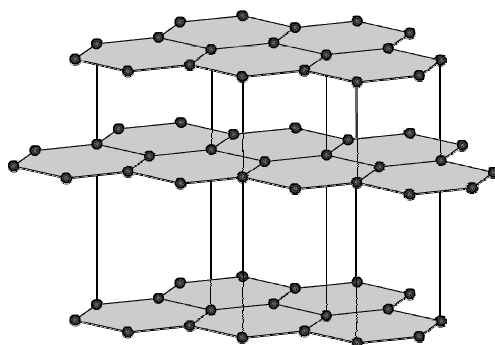


Figure 3.4 An example crystallized graphite growth [91].

- Atomistic Lattice The atomistic lattice model simply has each mathematical point representing each atom within a protein [79]. For example, illustrated in Figure 3.4 is a small model of crystallized graphite. If the research uses an atomistic lattice model, each point in Figure 3.4 becomes a point in the lattice space model.

- **Non-Atomistic Lattice** A non-atomistic lattice model is a bit more complicated. A single mathematical point can represent a group of atoms versus the atom per point model discussed above. For example, in Figure 3.4 it is easy to identify two separate surfaces. In addition to these two surfaces being constructed of the same type of atoms and having the same structure, they both can be grouped into two separate entities. These larger entities can then be treated as a single object with properties equal to that of all the atoms combined within the encompassing object.

These models can then be used in a quantum mechanical energy, molecular dynamic model, or Off-Lattice Monte Carlo simulators (described in Section 3.3.4) to find final resting conformations. It should be noted that lattice models come in many different configurations, like cubic, hexagon, and diamond shapes. These configurations fit nicely into scientific visualization techniques called grid alternative selection [81]. Furthermore, these models require grid generation for solving partial differential equations governing a model of some physical field phenomenon (in our case a protein model). This is usually executed in CAD data describing some geometry - specifically a set of finite points discretizing the given curves, surfaces, and possibly a surrounding volume might be the output. This technique is normally used when needing to evaluate a prototype's structural soundness and flight stability (vortexes) before huge amounts of money is invested in physically building the model; however, in this case a physical model is not of any value, but one can build the model virtually with these grids. These grids are decomposed into two types: *structured* and *unstructured* [81]. Structured grids refer to grids that are constructed of elements that are topologically equivalent to a square or cube. All other grids are termed unstructured. In Figure 3.5, there are six grids shown. The PSP problem is mapped onto an unstructured grid if using the Atomistic or Non-Atomistic Lattice model. However, upon mapping a protein to a grid resolution can be impacting the solution resolution the new grid model might be able to achieve.

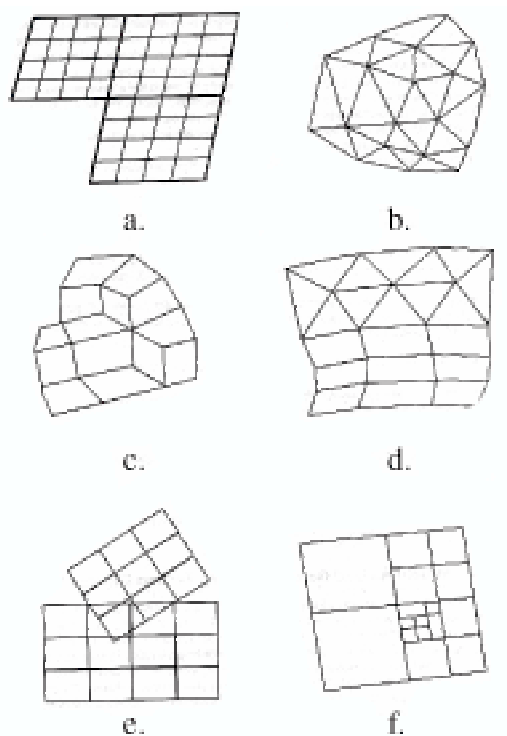


Figure 3.5 Six grid alternative selection configurations are shown: a) multiblock structured, b) unstructured-triangular, c) unstructured-quadrilateral, d) hybrid, e) Chimera, and f) hierarchical grid [81].

3.3.2 Molecular dynamic model simulation. Molecular dynamic (MD) model simulation is used as another approach to simulate protein folding. This approach focuses on the known properties of a protein, its atoms, bonds, and the physical world. The simulator is programmed to have each atom interact as they would in the real world - hence, a simulation of the entire folding process results. Because each atom has numerous intensive calculations to determine attraction/repulsion, positioning, bond flexibilities, etc, this method can only handle small proteins and takes an unreasonable amount of time to converge to an answer. Furthermore, this is exactly why IBM has determined that a dedicated computer architecture is needed in simulate using MD models [23]; however, as discussed in section 3.3, IBM still has limitations on protein sizes even with a petaflop computer. This technique might be more useful if we were given a conformation that is close to the correct conforma-

1	Select a point ^a in phase space ^b , x (initialization)
2	Create a new state from x , x' (Markov process ^c)
3	Compute the transition probability of state $x \rightarrow x'$, $W(x,x')$
4	Generate a (pseudo) random number, R , uniformly distributed in $[0,1]$
5	If $W < R$ then state is unchanged, remains as x .
6	Repeat (from step 2). Otherwise, accept x' as the new state and repeat (from step 2).

^aA point in the algorithm could be either a group of atoms' or a single atom's position

^bPhase space could be selected subspace within the entire area (select few atoms) to optimize or it could be the entire space itself (all atoms)

^cA first order Markovian process is when a random event or next event is dependant on only the most recent observation [92].

Table 3.2 General algorithm for a Monte Carlo simulation.

tion of the protein, we could then submit the 'close' answer to a molecular dynamic simulator and possibly get the answer more quickly than if the simulator were given no conformation information at all. This concept is equivalent to using a localized search or *fine tuning* on thought to be close conformations.

3.3.3 Monte Carlo. Monte Carlo simulations are based on the the evaluation of a system by generating random solutions to solve problems of any kind. This is true for the PSP problem as well, in fact, there are are many examples of such work [83].

The general algorithm for a Monte Carlo method [45] is in Table 3.2.

The Monte Carlo algorithm in Table 3.2 displaces atoms, one at a time, until the overall energy or conformation evaluates to something with better merit. This randomness might provide good solutions; however, because it is a memoryless method, it may prevent previously good partial conformations from evolving into the correct final conformation.

3.3.4 Off-lattice Monte-Carlo Simulations. A different kind of simulation method called off-lattice simulation is close to a divide and conquer algorithm. It calls for the drawing boxes around parts of the protein structure to be evaluated.

This prepares the protein's initial configuration by drawing a desired density (box size and number of particles) about the area to conform. Normally, a simple lattice cube is utilized for the density box. Within each density box, atoms are selected one at a time and moved a random distance from their present location. Figure 3.6 illustrates density boxes and atom selection and agitation. Energy calculations are performed before and after the move. If the energy value is lowered due to the move, the new atom position is kept; otherwise, the atom is moved back to its original position. This continues until results are good enough or supplied stopping criteria is met. In Figure 3.6 the darkened circles represent the atoms within the active density box where atoms are chosen. The arrows represent the forces from atoms that are close enough to interact with selected atoms within that density box. [60]

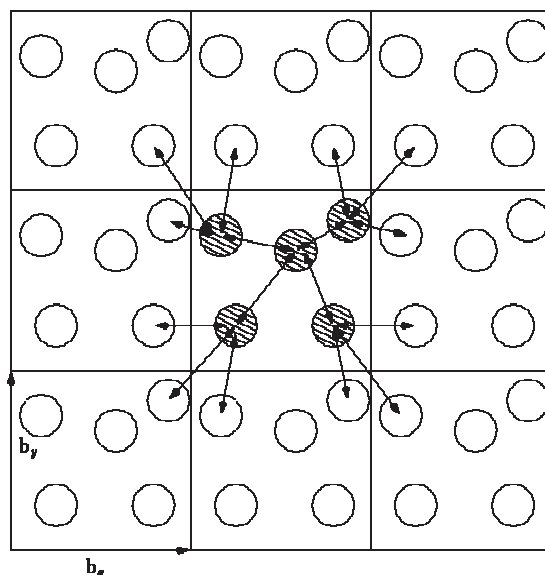


Figure 3.6 Illustrating the density boxes and analysis of atoms within each box during the Off-Lattice Simulation method [60].

3.4 Energy Minimization Methods

The energy landscape algorithms are based on the idea that a protein's final resting conformation is found at the conformation that yields the lowest overall energy of the protein. Force field energy equations, like Assisted Model Building with

Energy Refinement (AMBER) [58], Empirical Conformational Energy Program for Peptides (ECEPP) [87] and CHARMM, are typically used in calculating the energy for a specific conformation. The disadvantage to using these methods are two fold: 1) the problem of enumerating every possible conformation a protein could possible retain and 2) a highly computational fitness or energy evaluation function that needs to be evaluated at each of the possible conformations. Given by the fact that the number of conformations a protein could retain is uncountably infinite, it is impossible to enumerate every possible combination of conformations. Even when decomposing a protein into dihedral angles and limiting the values that theses dihedral angles can take on the number of possible conformations would be $(Number_of_Values_a_Dihedral_can_have)^{Number_of_dihedral_angles}$. Furthermore, it has been measured that one fitness evaluation takes 15msec on today's high-end computers, it can be concluded that the fitness evaluations of each conformation alone for even small proteins, is going to cost much computational time (approximately $2.5 * 10^{114}$ msec from Equation 2.4). This is precisely why alternative algorithms have evolved to solve specific problems. Following is a discussion of GA's employed at AFIT for solving the PSP problem [90] [56] [80] [84]. The PSP problem has been attacked with many different forms of GAs including: simple GA (sGA), mGA, Parallel GA (PGA), fmGA, pfmGA, GAs with local searches (Lamarckian, Baldwinian, and Niching), and other smart operator techniques. All the following GAs utilize the same CHARMM energy model as a fitness function. This fitness function was ported from Fortran to C as the choice fitness function in 1991 [35]. A short discussion of fitness functions and operators to applied to these function is found in Chapter 2.

3.5 *Historical Perspective and AFIT GAs*

In 1991 AFIT launched an effort to solve the PSP problem using a GA approach. Captain Laurence Merkle applied both a sGA and mGA (Defined below)

[68]. GAs are "search algorithms based on the mechanics of natural selection and natural genetics" [40]. These algorithms are rooted back to the Darwin's theory of evolution, natural selection, and genetics [46]. All GAs discussed within this Thesis have originated from the sGA. Furthermore, understanding the sGA is paramount to understanding such extensions.

The first step to applying a GA is to transform the problem domain solution variable structure into a fixed length binary string - called chromosomes. In other words, a solution should be representable by one chromosome. Individual elements of a chromosome are called features - corresponding to the genes of a chromosome. Feature values are the values that one feature may take on - these represent alleles of a gene. The set of every allele is the genetic alphabet [40]. An example of a genetic alphabet is the set 0,1 and 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F - better know as binary and hexadecimal alphabets respectively. After a discretized encoding scheme is applied to the problem, there must be a way to decode and evaluate the merit of a specific chromosome or solution. This is normally called the fitness function - it checks the fitness or merit of a solution. Its main purpose is to give an indicator if one chromosome is better than another. To evaluate the fitness of a chromosome, a decode occurs. Then a the fitness evaluation concludes which is a high computational analysis of the chromosome. This costs the algorithm in time (Such as the energy fitness function in our GA used to search the PSP problem energy landscape).

3.5.1 Simple GA. The main routine in a sGA, after encoding the problem, builds a population of chromosomes. It then selects an individual from the current population and uses reproduction via crossover and mutation to generate a new population - each time evaluating the newly created chromosome's fitness. Each new population member is placed in the a new population pool. When the new population pool is full (population size is predetermined), then the new population replaces the old. This is a generational GA. The routine then repeats itself. Figure 3.7 illustrates this cycle. The dotted lines indicate the barrier between the real solution problem

domain and the encoded solution or chromosome domain. Complexity estimates for the Simple GA are in Table 3.3.

<i>Phase</i>	<i>Serial</i>
Initialization	$O(l^n)$
Recombination	$O(g*n*q)$
Exchange Populations	$O(1)$
Overall mGA	$O(l^n)$

Table 3.3 Complexity Estimates for the sGA. Where l is the length of chromosome, n is the size of population, q is the group size for a tournament selection and g is the number of generations.

3.5.2 Steady State GA. The main routine in a Steady State GA (ssGA), after encoding the problem, builds a population of chromosomes. It then selects an individual from the current population and uses reproduction, crossover and mutation to generate new population members - each time evaluating the newly created chromosome's fitness. Upon evaluation of a better chromosome, that better chromosome is placed into the original population. The routine then repeats itself. Figure 3.7 illustrates this cycle. The dotted lines indicate the barrier between the real solution problem domain and the encoded solution or chromosome domain. Complexity estimates for the ssGA are in Table 3.4.

<i>Phase</i>	<i>Serial</i>
Initialization	$O(l^n)$
Recombination	$O(g)$
Overall mGA	$O(l^n)$

Table 3.4 Complexity Estimates for the ssGA. Where l is the length of chromosome, n is the size of population and g is the number of generations of reproduction.

Normally, the initial population of chromosomes is randomly generated in an effort to give good exploration of the search space. Additionally, the size of the population is maintained at an "optimal number" to help aid selective pressure during the progressing search. This optimal number can be described by the schema

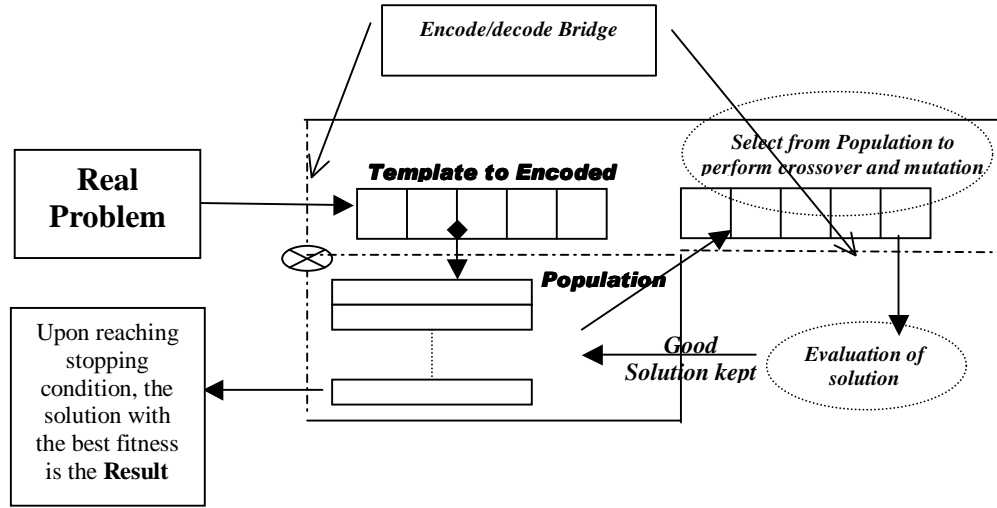


Figure 3.7 High Level overview of solving real world problem with a sGA.

theorem where the proportion of selection is used in determining what is needed to increase the probability of instances of a schema occupy the population [3]. This is important because large population sizes result in large memory space usage, stressing and degrading the efficiency of the algorithm. Also, under specifying a population size causes poor effective algorithm results. In addition to population sizing, another interesting part of GAs is the application of different operators to find better chromosomes. Basic operators come in the form of crossover and mutation. These operators are applied after a selection mechanism. Other operators have been used specifically to improve upon solutions for the PSP problem [7]; however, none have proved to be dominate. The mutation operator, crossover operator, and choice selection mechanisms are defined as the following:

- Selection

Reproduction begins with the selection mechanism. The following are the two main selection pool determinations for the selection mechanism.

1. μ represents parent solutions and λ represents offspring solutions.
2. (μ, λ) Offspring is chosen from the offspring solutions only. The offspring is generated by using mutation and crossover operators.

3. $(\mu + \lambda)$ Offspring is chosen from the parents and offspring solutions both.

The offspring is generated by using mutation and crossover operators, but the parents are acquired from the previous population.

Once the selection pool is determined, then the selection mechanism itself must be chosen. The idea is to choose a mechanism for developing the speciation of the population into something that results in finding good solutions. There are many selection mechanisms. The following is a discussion of only two: roulette wheel and tournament selection. The fmGA uses tournament selection with thresholding as its selection mechanism; therefore, these are defined as the following:

1. Roulette Wheel

Selection using this type of mechanism is common. Each member is assigned a slice of a pie. Where the entire pie represents the total fitness of every member in the population and each member's slice is the proportion of their fitness with respect to the total of the entire population's fitness (See Equation 3.2). Once all member have been assigned their slice, a random number from 0 to the sum of every population member's fitness is generated to indicate which member has been chosen. This gives the members with better fitness values a higher probability of being selected.

[3]

$$\forall_j \exists Slice_j = \frac{f(m_j)}{\sum_{i=1}^n f(m_i)} \quad (3.2)$$

Where n is the size of population and m is a single member.

2. Tournament Selection

This is another common selection method. A group of q populations members is randomly selected from the population. They can be selected with replacement or without. This group takes place in a tournament

and the winner is determined by its fitness value. q is called the tournament size. The winner is placed in the new population pool and the process is repeated until the next population is full. [3]

Thresholding

Thresholding is a constraint added to the selection of the q tournament members. Each member is selected a certain predefined distance apart from one another. This is essential to prevent incest among population members. This is also used in the fmGA used in this Thesis. [3]

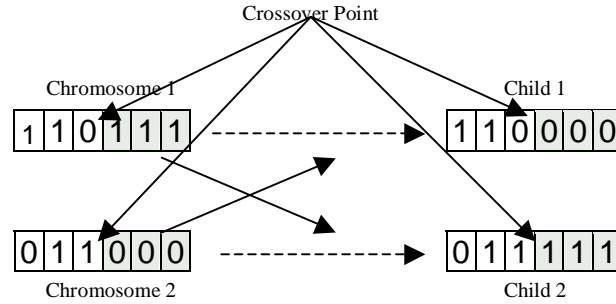


Figure 3.8 An example of single point crossover applied after selecting two population members.

- Crossover

This is the primary device for the fmGA to reproduce. Its basic purpose is to take two chromosomes from the population, cut them between the same two genes and splice the 1st half of the 1st chromosome with the 2nd half of the 2nd chromosome and the 2nd half of the 1st chromosome with the 1st half of the 2nd chromosome. Thereby making two new chromosomes ready for evaluation. This is represented by example in Figure 3.8. The example illustrates a single point crossover; however, crossover may be utilized as a multi-point operator where only a subsection, identified by two or more points, of the chromosome is crossed over. [3]

- Mutation

The mutation operator is needed for increasing exploration by reducing selec-

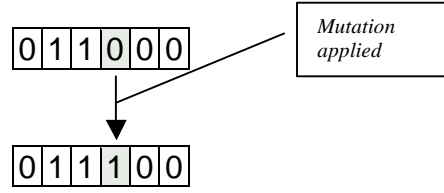


Figure 3.9 A simple example of mutation applied.

tive pressure during the search. It is normally employed only a small percentage of the time. This operator's application occurs by changing a randomly selected gene. For canonical GAs, it simply flips the bit if the alphabet consists of 1s and 0s. Figure 3.9 uses an example to illustrate how this operator is applied. This type of mutation can be called hill climbing or sweeping. For real valued GAs genes are shuffled around, like the *scramble* mutation operator [95][3].

Not all variations of crossover and mutation nor are all GA operators discussed here; however, those covered are integrated in the fmGA used in this Thesis. Other operators used in other GAs are the following: transposition, translocation, conjugation, inversion, transduction, gametogenesis, transcription and translation. A discussion of how these operators relate to biology can be found in Appendix F.

3.5.3 Messy GA. The mGA was also implemented to solve the PSP problem by Merkle [68]. The original mGA was designed specifically to solve deceptive problems - problems where the sGA and ssGA get caught in suboptimal trenches in the fitness landscape without hope of climbing out [10]. These types of problems are called deceptive because there is no path from one semi-good set of solutions to the optimal solution. For example, a deceptive problem might be a binary string of ten bits represented the problem and good semi-optimal answers could be found at sequences of bits with only a single 1 in the string. But the best or optimal solution is found at the string sequence with all the bits being 1 [68]. Furthermore, because combinations of other sequences of bits (combinations with two or more, but not all bits are 1) yield poor fitness evaluation values, the sGA is kept from finding the

optimal answer. This brings to focus that crossover and mutation may not be flexible enough to find optimal solutions to deceptive problems causing the need for the mGA.

<i>Phase</i>	<i>Serial</i>
Initialization	$O(l^k)$
Primordial	0
Juxtapositional	$O(l \log l)$
Overall mGA	$O(l^k)$

Table 3.5 Complexity Estimates for the Original mGA [42]

The mGA consists of initialization, primordial and juxtaposition phases. The use of partially enumerative initialization allows the mGA to find optimal solutions to deceptive problems such as the one described. The partially enumerative initialization builds a population of Building blocks (BB) consisting of all possible partial solutions of a specified length. Furthermore, if the BB size is equal to or greater than the deception present, then the initial population contains parts of the optimal solution before the search begins and the probability that the GA finds the optimal answer is increased. See Appendix G for a discussion of building blocks and their association to finding good solutions.

The differences between a mGA and a sGA are many! The initial populations are much different. The mGA produces a population of partial solutions (BBs of a specified length) whereas the population of the sGA is a group of chromosomes or complete solutions. The fitness function for the mGA is modified (uses a competitive template (CT)) to handle partial solutions by being able to evaluate BB size solutions; on the other hand, a sGA can only evaluate an entire chromosome's fitness for comparison. Associated with the type of population members held in the population is the population size itself. The mGA has a larger initial population size than that of a sGA. The mGA's population size can be calculated using Equation 3.3 where k is the block size, l is the length of the string, and c is the cardinality [33]. The idea for such a large population size is to have every combination of a particular

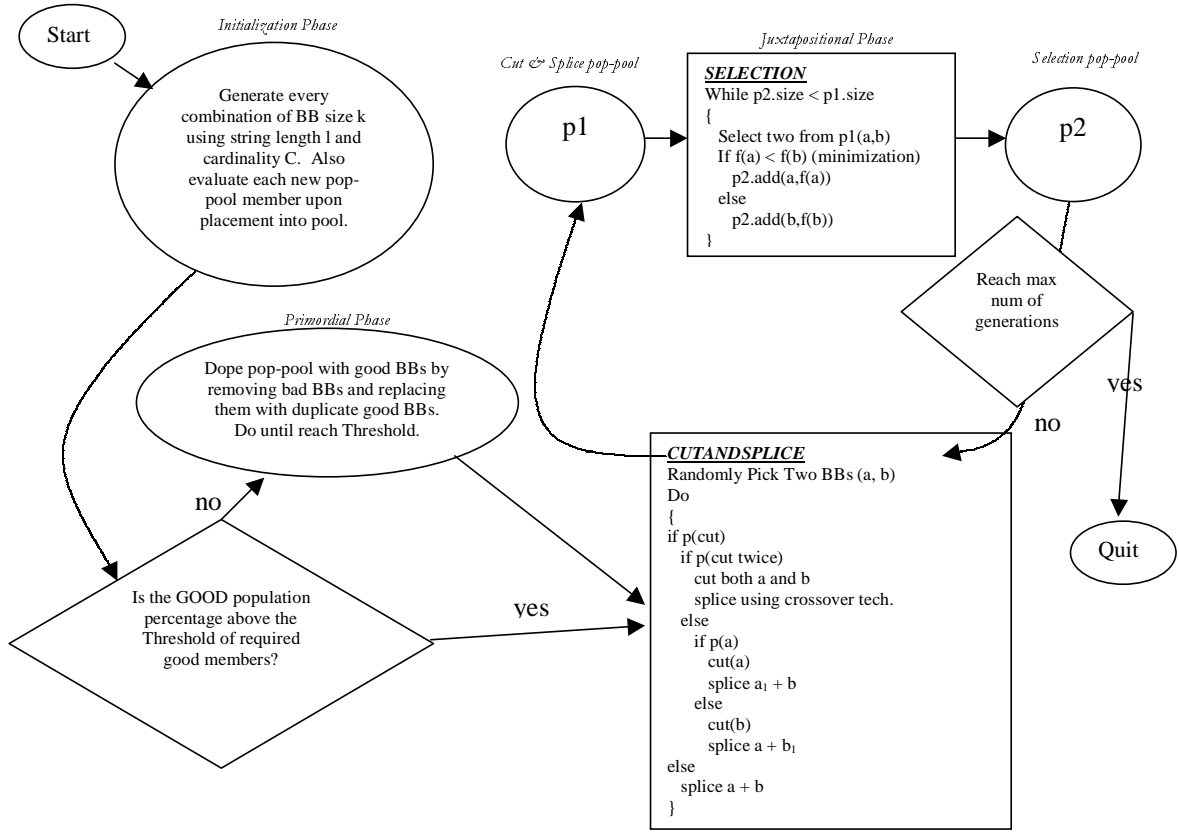


Figure 3.10 Program flow of the mGA [70]. The complexity for this algorithm can be found in Table 3.5

block size to be present for the insurance of having the answer to the problem held in the population. Furthermore, the mGA enriches the pop-pool with good building blocks in some cases allowing duplicate strings to reside and periodically reduces the total population size during selection. Finally, the mGA has variable length strings maintained in the population where the sGA population members are essentially always the same length - that of the original fixed string length.

$$k^c * \binom{l}{k} \quad (3.3)$$

Finally, these two GAs are similar in having the objective to obtain population members that help in yielding good generations - ultimately finding semi-optimal

<i>Phase</i>	<i>Serial</i>
Initialization	$O(l)$
Primordial (Building Block Filtering)	$O(l^2)$
Juxtapositional	$O(l \log l)$
Overall fmGA	$O(l^2)$

Table 3.6 Complexity Estimates for the fmGA [42]

solutions. Figure 3.10 is a flow chart that describes the program flow of the mGA. The only item left out is how the fitness is evaluated each time a member is added to any pop-pool. When the mGA initializes, it builds a CT by randomly generating a string of the same size as a solution. Partial solution fitness is evaluated by substituting the partial solution into the CT – replacing CT bits with bits found in the partial solution, then the entire new CT is evaluated. Once finished, the entire new CT is destroyed unless it is found to be better than the best competitive template found (compared to the best found CT). If the new CT is the best found, it is stored as the best found and at the end of each generation run the CT is replaced with the best found CT.

The mGA solves a deceptive problem by creating a population size that contains every possible combination of a particular block size; therefore, also containing the solution. This effective algorithm comes at cost in complexity (See Table 3.5 for mGA’s complexity), execution time, and memory space.

3.6 *fast messy GA*

After the sGA and mGA research continued with the studying the effects of a paralleled version of both GAs in 1992 [9]. Following this, the fmGA was to be named as the GA of choice [32] in 1993. Much has been written about the PSP problem using the fmGA. [34], [32], [69], [76], [77], [18], [17], [16] and [19] scoped the improvement of using the fmGA. Currently, we use this algorithm as the primary search engine for solving the PSP problem. Additionally, we use this algorithm

The mGA's advantage over the sGA is its ability to explicitly create tightly linked building blocks for the optimization of deception problem - basically, defeating deception by insuring that it has the answer somewhere in the population of building blocks it creates in the initialization phase. However, the mGA's insurance policy does not come at cost; indeed, it is extremely expensive to build every combination of a particular building block size to put into a population. This initialization dominates the entire algorithm [41]. The fmGA is designed to reduce this complexity by replacing the initialization phase and primordial phase with a probabilistic complete initialization (PCI) and primordial phase consisting of selection and building block filtering (BBF). PCI and BBF are an alternate means to providing the juxtaposition phase with highly fit building blocks [42]. The entire program flow for the fmGA is illustrated in Figure 3.11. When comparing the complexity of the fmGA (Table 3.6) to that of the mGA (Table 3.5), it can be concluded that the fmGA has lower complexity.

$$2^k \frac{\binom{l}{l'}}{\binom{l-k}{l'-k}} 2c(\alpha)\beta^2(m-1) \quad (3.4)$$

The PCI phase creates an initial pop-pool size of n described by Equation 3.4, which is probabilistically equivalent to the pop-pool size at the end of the primordial phase of mGAs.

$$p(l', k, l) = \frac{\binom{l-k}{l'-k}}{\binom{l}{l'}} \quad (3.5)$$

It is accepted as true that the population size is the multiplication of three equations: The gene-wise probability equation, the allele-wise combinatoric equation, and the building block evaluation noise equation [42]. Furthermore, it can be shown that the probability gene-wise equation is the probability of selecting a gene combination of size k in a string of length l' having the total number of genes, l , is given as Equation 3.5. If, m_g , is assigned to the *inverse* of Equation 3.5, it is sug-

gested that each subpopulation of size n_g have one needed string, on average, gene combination of size k . Equation 3.6 defines n_g . If this equation suggests that we expect to have one of our needed gene combinations for one particular building block size k , then we can further say that we want to have the needed gene combination for each and every possible combination of k building block size, which makes for 2^k allelic combinations or allele-wise combinatoric population size multiplier. A second multiplier is then defined in Equation 3.7 called the building block evaluation noise equation. This equation makes for a population size calculation where the selection error between two competing building blocks is no more than an α different. Finally, we have a simple, more manageable, population sizing calculation Equation 3.8. [42]

$$n_g = \frac{1}{\frac{\binom{l-k}{l'-k}}{\binom{l}{l'}}} \quad (3.6)$$

$$n_a = 2c(\alpha)\beta^2(m-1) \quad (3.7)$$

$$n = n_a n_g \quad (3.8)$$

Once the population size is determined, the initial population is created and the algorithm begins. The length of strings, l' , is set to $l - k$. The primordial phase performs several tournament selection generations to build up copies of highly fit strings followed by BBF to reduce the string length toward the building block size k . See Figure 3.11 for the program flow. It should be noted that building block filtering is nothing more than randomly deleting genes from a particular string - effectively reducing it [32]. An example of population sizing calculation is shown in Figure 3.12. Two 3D plots in Figure 3.13 have been generated to illustrate how the fmGA consistently generates smaller population sizes. Observe that the fmGA and mGA have the same juxtaposition phase. Please refer to Figure 3.11 for the flow of this

g := 0
a := 1
mga := 2
popsize := 3
n_a := 50

A typical population size calculated for the fmGA versus mGA. Variables set to typical numbers found for a run using the MET protein.

l := 240
k := 16
l' := l - k
l' = 224

$$p(l', k, l) := \frac{\text{combin}(l - k, l' - k)}{\text{combin}(l, l')}$$
 $p(l', k, l) = 0.319$ probability a gene exists for k

$$n_g := \frac{1}{p(l', k, l)}$$
 $n_g = 3.131$ gene-wise, allele-wise

n_a := 10

$$n_{\text{popsize}} := n_g \cdot n_a$$

$$n_{\text{mga}} := 2^k \text{combin}(l, k)$$

Population Size for fmGA = $n_{\text{popsize}} = 31.308$
Population Size for mGA = $n_{\text{mga}} = 2.276 \times 10^{29}$

Population size differences for the two algorithms is astonishing. Population size directly impacts algorithm run time and resource requirements.

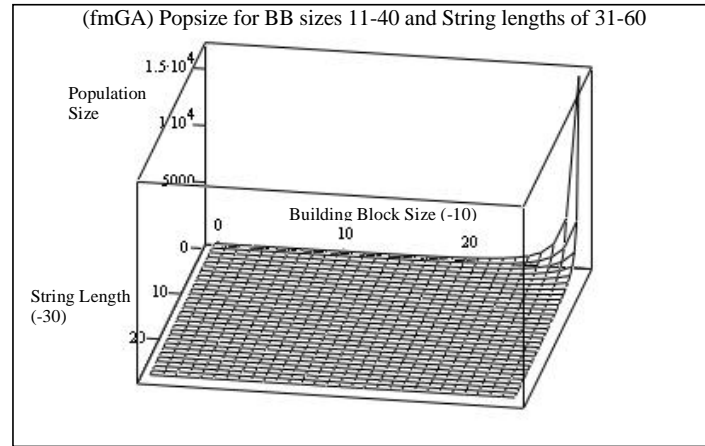
Figure 3.12 An example of typical calculations to find population sizes for the fmGA and mGA. It should be noted that the population size change for the fmGA as the building block size change throughout the algorithm.

phase and the flow of the entire algorithm. To conclude, instead of having a huge initialization cost as we do with the mGA, the fmGA has allowed a more optimal initial population mechanism that is statistically equivalent to that of the mGA.

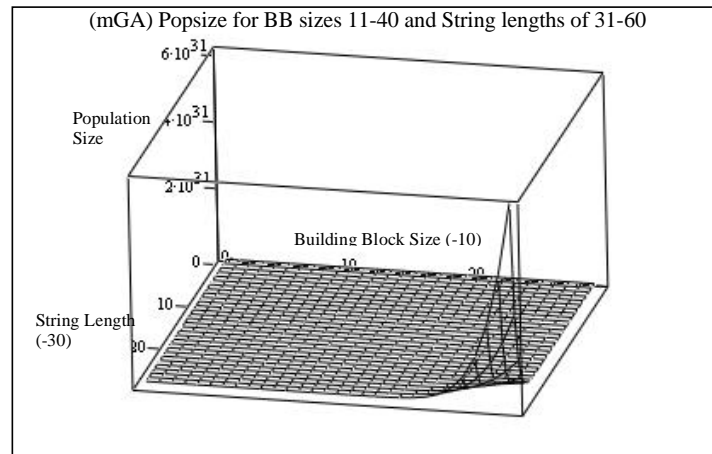
This concludes the discussion of previously applied GAs to solving the PSP problem at AFIT. The fmGA was followed by the following GAs: Combined Algorithm [35], Hybridized GA [35], Real Value GA [50], and a Linkage Learning GA [20]. These algorithms are discussed in Appendix H.

3.7 Parallel fast messy GA

The pfmGA is an extension of the fmGA [42] and is a binary, population based, stochastic approach that exploits BBs within the population to find solutions to



fmGA_popsize



mGA_popsize

Figure 3.13 The upper plot is of the fmGA calculated population sizes and the lower plot is that of mGA population sizes. The fmGA consistently produces population sizes orders of magnitude lower than that of the mGA. Population sizes are on the z axis while the x and y are reflecting indexes to building block sizes of 15-75 and string lengths of 20-80.

optimization problems. Our pfmGA may be executed in a single program single data (SPSD) or a single program multiple data (SPMD) mode. The parallelization of this algorithm is based on the Message Passing Interface (MPI) constructs. The pfmGA consists of three phases of operation: the Initialization, Building Block Filtering, and Juxtapositional Phases, all using synchronous MPI based communications [77]. The pfmGA operates independently on each of the processors with communications

occurring during the Initialization and Juxtapositional phases, this is referred to as the Independent mode.

In the Initialization phase, a population of individuals is randomly generated on each processor. The size of the population is based on an equation calculated from the string length of the population members, the user specified BB sizes *a priori* chosen, and the number of processors available. Subsequently the population members are evaluated. A Competitive Template (CT) is also generated on each processor. The CT is a locally optimized population member necessary for calculating the fitness value of population members in the later phases of the algorithm. After a local optimization of the templates is conducted on each processor, the best found template becomes the new template on each processor.

The Building Block Filtering (BBF) Phase follows and extracts the BBs from the population for manipulation and the generation of solutions. This process occurs through a random deletion of bits from each of the population members alternated with tournament selection. A BBF schedule is provided *a priori* to specify the generations for the deletion to occur, the number of bits to be deleted from each population member and the generations to complete tournament selection. This phase completes once the length of the population members' chromosomes have been reduced to a predetermined BB size. In order to evaluate these BBs ("under-specified" strings), throughout the phase a competitive template is utilized to fill in the missing allele values. These population members are referred to as "under-specified" since each locus position does not have an associated allele value. The BBF process is alternated with tournament selection to keep only the strings with the best building blocks found, or those with the best fitness value around for later processing.

The Juxtapositional phase follows and uses the building blocks found through the BBF phase and recombination operators to create population members that become fully specified (all loci values have corresponding allele values) by the end

of the phase. Again the competitive template is used anytime a population member is missing a locus and in the case of “over-specification”, where a specific locus is assigned an allelic value multiple times, the first value encountered is the one recorded. At the end of the Juxtapositional phase, the best population member found across all of the processors becomes the new competitive template on each processor. At this point the BB size is incremented and each of the three phases are executed again. After all of the specified BB sizes are executed, the best solution found is recorded and presented to the user.

3.8 *Multi Objective fmGA (MOfmGA)*

The MOfmGA executes using the same basic algorithm structure as the fmGA. The differences are slight. First, the MOfmGA automatically uses a multiple competitive template design where each objective function is *assigned* a competitive template. This competitive template evolves to “optimize” that particular objective function. Each population member is overlayed onto this competitive template before evaluation of this objective function. Secondly, as the Juxtapositional Phase completes, population members (after overlaying onto a competitive template if necessary) are written to a file for post mortem processing and extraction of pareto front points. Finally, after storing the overall best chromosome into the being the next competitive template, a PDB file is generated using the structure having the overall best fitness. This file is used for post mortem viewing of the structure after completion or during execution of the program.

3.9 *Basic evolutionary algorithmic approach justified*

Many EAs can be applied to the PSP problem. Certainly, Evolutionary Strategies, Evolutionary Programming, Genetic Algorithms, and Genetic Programming could all be used. However, we have a working copy of the fmGA that also has an

integrated CHARMM model. A discussion of all these alternate approaches can be found in Appendix I.

3.10 Simplified Mapping Problem Domain to Algorithm Domain

The mathematical model for a protein conformation is rather complex and can be represented in different forms depending on the algorithm to solve the problem. This is the formalization used in this thesis effort.

3.10.1 Mathematical/Symbolic model.

- A = the set of amino acids (amino acid sequence) = $\{a_1, a_2, \dots, a_n\}$. For example MET = $\{a_1, a_2, a_2, a_3, a_4\}$ where 1=Tyr 2=Gly 3=Phe 4=Met
- a_i = the set of atoms that make-up a particular amino acid i . ($i \in \{1, 2, \dots, 20\}$) which represent the 20 different amino acids.
- Aa([Amino Acid]) - function that breaks the amino acid up into a sequence of atoms. Additionally each atom is given a cartesian coordinate (x,y,z) to specify where the atom is in 3D space.
- Da([List of Atoms]) - function that distributes a list of atoms into dihedral angles which specifies every atom with a set of four connected atoms.
- aD([List of Dihedral angles]) - function that takes the dihedral angles and back calculates the cartesian coordinates (x,y,z) of each atom.
- D - the set of dihedral angles which fully specify every atom in a protein. Additionally, D is the search variable. $D = \{d_1, d_2, \dots, d_n\}$
- Protein, P, is specified in amino acids sequence.

$D = Da(Aa(P))$ Ready program to interrogate dihedral angles.

$\sim D = operation(D)$ Algorithm changes dihedral angles.

Evaluate(aD($\sim D$)) Determine the merit of the change.

3.11 Problem/Algorithm Domain Operational Design Specification form

From this point forward there could be two different search techniques employed: Deterministic (Depth First Search with backtracking (DFS/BT)) and Stochastic GA. It is important to mention that using either of these search techniques still requires the trimming of the search space discussed earlier; moreover, the deterministic search requires a much larger amount of time than the GA in finding good solutions to this problem regardless of the protein size. Therefore, a GA should be employed in this thesis effort. It should be mentioned that there are approaches considered to be a cross between a deterministic and Stochastic methodologies using nonlinear optimization techniques called a maximum likelihood approach [24]. However, only the deterministic and Stochastic GA are described using the following algorithm domain mapping.

- Initialization

$P_i^\sim = M_i(P_i)$ The Set of atoms in protein P_i is parsed for matching amino acids. Atoms making up the amino acids are given x-y-z values known a priori (using pre-canned amino acid structure coordinates). It is important to note that x-y-z coordinates are given WRT the last atoms, so there is no need to keep track of where each atom is on a 3D grid.

$P_i = I(P_i)$ All bond lengths and angles between three atoms distances are set to ideal.

- Output Domain $P_i = a_1, a_2, \dots, a_t$ The final or the latest conformation having the best fitness found

$P_i^\sim =$ Where a_j contains the spherical coordinates connecting atoms outside amino acids. $a_j = (\rho, \tau, v)$: ρ is equal to the ideal bond distance. j is equal to the atom i within P_i .

- Operators $T(a_i) \equiv$ The transformation of a_i 's spherical coordinates into x-y-z coordinates stored in P_i .

$D(a_i, a_{i+1}, a_{i+2}, a_{i+3}) \equiv$ Calculating a dihedral angle given 4 atoms ($j = 0, 1$).

$$u_j = (a_{i+2+j}.z - a_{i+j}.z) * (a_{i+1+j}.y - a_{i+j}.y) - (a_{i+1+j}.z - a_{i+j}.z) * (a_{i+2+j}.y - a_{i+j}.y) \quad (3.9)$$

$$w_j = (a_{i+1+j}.z - a_{i+j}.z) * (a_{i+2+j}.y - a_{i+j}.y) - (a_{i+2+j}.z - a_{i+j}.z) * (a_{i+1+j}.y - a_{i+j}.y) \quad (3.10)$$

$$g_j = (a_{i+2+j}.y - a_{i+j}.y) * (a_{i+1+j}.x - a_{i+j}.x) - (a_{i+1+j}.y - a_{i+j}.y) * (a_{i+2+j}.x - a_{i+j}.x) \quad (3.11)$$

$$N_j \equiv u_j \hat{i} + w_j \hat{j} + g_j \hat{k} = u_j a_{i+j}.x + w_j a_{i+j}.y + g_j a_{i+j}.z \quad (3.12)$$

$$\therefore d_\theta = \frac{N_0 \cdot N_1}{|N_0||N_1|} \quad \text{where} \quad |N_j| = \sqrt{u_j^2 + w_j^2 + g_j^2} \quad (3.13)$$

Dihedral angle is described by Equation 3.14. Note that the sign of the Dihedral angle is dependent on the sign of the cross product of the two planes N_0 and N_1 .

$$\therefore \theta_{RAD} = \frac{N_0 \times N_1}{|N_0 \times N_1|} \cos d_\theta \quad (3.14)$$

$$\text{Let} \quad a = B_{L(i)} \quad \text{and} \quad b = B_{L(i-1)} \quad (3.15)$$

$$\therefore c = \sqrt{x_i^2 + \mu_i^2} \quad \implies \quad c = \sqrt{(x_{i-2} - x_i)^2 + (y_{i-2} - y_i)^2 + (z_{i-2} - z_i)^2} \quad (3.16)$$

The bond angle, μ , is described by Equation 3.17 and bond lengths are represented by B_L in Equation 3.15. Furthermore, it is easy to find bond lengths using the distance equation from calculation shown in Equation 3.16.

$$\therefore \mu = \cos\left(\frac{a^2 + b^2 + c^2}{2ab}\right) \quad (3.17)$$

3.11.1 Extended Algorithm Domain.

$A = (A_1, A_2, \dots, A_m)$ where atoms are sorted according to position in a protein

$B = (b_{ij}, b_{ij}, \dots, b_{ij})$ where i and j are the atoms bonded together

$P_j^\sim = (a_3, a_6, \dots, a_q)$ where a_i is a mate to the atoms in A – updates

to these atoms directly updates corresponding atoms in set A

$\forall h \forall k | B_k(i = h) \cap (A_j \text{ WRT } A_k \text{ define (previous atom) from origin}$

$\forall h \forall k | B_k(j = h) \cap (A_i \text{ WRT } A_k \text{ for all atoms}$

3.12 Mapping Problem Domain to Algorithm Domain DFS/BT

Supposing that the search space might be manageable in size we could use the following deterministic dept-first search with back tracking algorithm to solve the PSP problem using fixed methodology. The algorithm is shown in Table 3.7.

3.13 Mapping Problem Domain to Algorithm Domain Stochastic

The PSP problem has yet to be solved with a polynomial running algorithm. Therefore, it is not in the class of P problems, but in NP and NPC problems. The search space for this problem is derived in Equation 2.4 after transforming it into a computer representable problem. In Equation 2.4 the search space is shown to be growing at an exponential rate as atoms are added.

A problem is in NP if it has a nondeterministic polynomial time solution; this means that the solution can be checked within polynomial time. If a problem is NP-complete, it means that a particular solution can be checked in polynomial time but to solve the whole problem (which often requires checking many possible solutions) requires an exponential time algorithm. Because an exponential function increases at a much more rapid rate than a polynomial, these problems are said to be intractable. For a (reasonable) problem size of 20, a polynomial algorithm might require $t \sim 20^m$ time steps, compared with $t \sim m^{20}$ for an exponential time algorithm). [57]

1	Step (0) Initialization
2	Set all angles in P_j^\sim to the lowest possible valid angle.
3	Step (1) Evaluate
4	If $f(P) < f(best_found)$ then
5	$Best_found = P$
6	EnD
7	Step (2) Stopping Condition
8	If angles of P are all set to Maximum valid angles
9	stop
10	End
11	$x = 0$
12	Step (3) Move Forward (1st angle)
13	Increment angles ψ and θ of each a_i in P_j^\sim
14	(This is done in a gear like manner)
15	Increment θ_x
16	If $Max_Allowable(\theta_x) + 1$
17	Goto Step (5)
18	End
19	Step (4)
20	Goto step (1)
21	Step (5) Move Forward (2nd angle)
22	$\theta_x = 0$
23	$\psi = \psi_x + 1$
24	If $Max_Allowable(\theta_x) + 1$
25	If $x = A $
26	Stop (search is finished)
27	End
28	$\theta_x = 0$
29	$x = x + 1$
30	Goto Step (1)
31	End
32	Goto Step (1)

Table 3.7 Pseudo code for DFS/BT Algorithm .

	Time	Quality of Solution
Deterministic	Does not Complete	Exact (Limited by constraints)
Stochastic	Reasonable (Tunable)	Semi-Optimal

Table 3.8 Comparison between Deterministic algorithm and Stochastic Algorithm applied to the PSP problem

Furthermore, it has been established that the combinatory nature of the protein structure would require an exponential time deterministic Algorithm to solve and is a NPC problem [57]

3.14 Comparison between mappings

To compare these two algorithms one must weigh the tradeoffs of each and select the tool according to the decision makers needs. In this case the problem has been transformed into a computer solvable problem and could be reduced to having a search space that is within the reach, time wise, of a deterministic search. However, if the search space is reduced, the resolution for solutions found is also decreased. This means that with a deterministic search algorithm the solutions found are not optimal and may not even be semi-optimal; however, the best solution with a constrained resolution is found. If instead of reducing the search space the resolution is increased and it goes beyond the capabilities, time wise, of a deterministic search, a stochastic search can be more effective in finding semi-optimal solutions. The algorithms are compared in Table 3.8.

3.15 Summary

This Chapter discussed the High to Low level design of the PSP problem being mapped to the fmGA algorithm domain. Background of previous AFIT and exterior departmental work have been covered in detail. Furthermore, alternate search tools have been discussed and justification has been given as to why this thesis work was accomplished using a fmGA. Chapter 4 develops the methodology

for the experiments. It covers the design of experiments and delineates the factors, metrics and statistical methods used in comparing solutions with previous work.

IV. Development Methodology

Discussed within this chapter is the design methodology of the MOfmGA. Development takes place in many stages. The first stage consists of understanding the original fmGA code. Small code modifications are made to gain knowledge of code stability, reproducibility of previous results, options and software engineering approaches. The first implementation of a multiple competitive template mechanism, the building block size study and the farming model is accomplished in this stage. The second stage focuses on a rewrite and the birth of a new algorithm, MOfmGA, and PDB file generation capabilities for structure visualization. The rewrite gave the MOfmGA the ability to solve multiobjective problems as well as keeping the enhancement to handle the generating and search development of multiple and different types of competitive templates. The final stage of design integrates Ramachandran Plots, problem domain information, as well as RMS cartesian coordinate and dihedral angle difference calculation into the MOfmGA. It was during this stage that final experimentation was accomplished and the investigation was complete. Normally, thesis experimentation is conducted on existing code; however, in this case, new code gives more functionality and flexibility to solving the PSP problem. Furthermore, this code allows for easier protein workload additions and result comparisons to accepted true conformations. Following is a discussion of the design effort undergone to build the MOfmGA for this thesis investigation.

4.1 Algorithm Design

Although the design changes for the algorithm are extensive, each experiment incrementally caused gradual modification to the existing algorithm until the multiobjective experiment. It is this experiment alone that caused almost an entire re-write of the algorithm. See Section 3.8 for a description of the MOfmGA. The following list of experiments motivated changes within the existing algorithm. Fol-

lowing this list, the design of each algorithm modification to support each experiment is discussed.

1. Multiple Competitive template
2. Farming Model
3. Protein 3D File Generation
4. Multiobjective
5. Ramachandran
6. RMS difference

4.2 Research Design

The fmGA is written in ansi C. Originally, it was written for use on Sun Workstations [32]; however, it has since been ported to run on Linux. Using Linux is advantageous for researches because it is an open source and free operating system. The algorithm had been changed significantly since its induction with Gates and, for the most part, had few Software Engineering practices followed during the last few years of transformation. In fact, the main function was several thousand lines long making for difficulty in understanding and adaptation. Design modifications made in support of experiments for this thesis investigation fit or modify the pseudo code for the fmGA found in [32].

4.2.1 Multiple Competitive Templates. Generation of multiple competitive templates during algorithm search is not a simple modification. As discussed in Section 3.8, each chromosome in the population must now have a fitness value associated with each competitive template maintained. Moreover, every evaluation call must evaluate a particular partial chromosome with every competitive template available. Pseudo code for this new evaluation mechanism can be found in Table 4.1.

```

# Passing any number “ct” lower than the total number of competitive
# templates results in the evaluation of only that competitive template
# for a specific chromosome. Passing a number equal to the total number
# of competitive templates results in an evaluation of all competitive
# templates. Passing a number greater than the total number of competitive
# templates results in an evaluation of the partial solution using the
# panmimetic competitive template.
1  Evaluation(c,ct)
2  partial_solution c
3  int ct
4  {
5      if (ct < max(ct))
6      {
7          c.f[ct] = charmm-eval(Overlay c → template[ct])
8      }
9      elseif (ct == max(ct))
10     {
11         for(j = 0;j < ct(max);j ++ )
12         {
13             c.f[j] = charmm-eval(Overlay c → template[j])
14         }
15     }
16     else
17     {
18         c.f[0] = charmm-eval(Overlay c → Panmimetic-template)
19     }
20 }

```

Table 4.1 Pseudo code for evaluation of partial solutions.

Furthermore, storing the best chromosome during algorithm execution and tournament selection must also be modified to account for having multiple competitive templates. Tournament selection is illustrated in Table 4.3.

In addition to having multiple competitive templates, the algorithm must be able to maintain a panmimetic (bitwise dominate) competitive template. This adds to having evaluation of not only each competitive template, but an extra evaluation of a panmimetic competitive template. Furthermore, all evaluations are increased by one evaluation and the operation for generating the competitive templates must include

Identifier	Sweeps	Evolving	CT Type	CT provided?	
CT chosen					
c	7	t	r	0	
Identifier	Obj 1	Obj 2	Obj 3	Obj 4	Obj 5
Multiobjective chosen					
m	1	2			
m	3	4	5		

Table 4.2 Competitive Template and Multiple Objective settings for the fmGA. This Table is describe more thoroughly in Appendix E.

a featured panmictic competitive template generation tool. The best stored is a rotating best (See Table 4.6), where the dominate bests are kept (See Table 4.5) – be they an ordinary or panmictic competitive template.

To initiate the detailed design changes, a extra file is generated to define the different competitive templates to be utilized. This file is illustrated in Table 4.2 and is more clearly specified in Appendix E. A competitive template is declared by placing a 'c' as the first single character on a line. For the algorithm to run, there must be at least one type of competitive template specified. For each competitive template identified the researcher must specify the number of sweeps, if it is to be an evolving competitive template, the type of competitive template to generate, and if the competitive template is being supplied. The number of sweeps identify how many times to sweep the competitive template upon generation. If a competitive template is non-evolving, it is set not to change during execution. Although other types of secondary structure exist – competitive template generation consists of the first two listed in Table 4.4 and random (r). Because POLY folds into an Alpha-helix structure the Alpha-helix was chosen as one model from which to build a competitive template. Furthermore, the Beta sheet was chosen to as a test for negative conformity. The final option allows the researcher to provide a competitive template within the option file. Each time a competitive template line is declared, a new competitive template is being added to the total number of competitive templates for that experiment (i.e. there is no limit to the possible number of competitive templates).

#	Passed are two population members c_1 and c_2 .
1	<i>Tournament</i> (c_1, c_2)
2	partial_solution c_1, c_2
3	{
4	$wins_0 = 0; wins_2 = 0$
5	for($j = 0; j < ct(max); j++$)
6	{
7	if ($c_1.f[j] < c_2.f[j]$)
8	{
9	$wins_1 = wins_1 + 1$
10	}
11	else if ($c_2.f[j] < c_1.f[j]$)
12	{
13	$wins_2 = wins_2 + 1$
14	}
15	# add nothing if it is a tie
16	}
17	i = RandInt(0,1)
18	if ($wins_2 < wins_1$)
19	{
20	return c_1 as winner
21	}
22	if ($wins_1 < wins_2$)
23	{
24	return c_2 as winner
25	}
26	else
27	{
28	switch(i)
29	case 0:
30	return c_1 as winner
31	break
32	case 1:
33	return c_2 as winner
34	break
35	}
36	}

Table 4.3 Pseudo code for tournament selection.

	Secondary Structures	Handedness	Residues per Turn
1	Alpha Helix (a)	Right	3.6
2	Beta Sheet (b)	Left Twist	
3	Beta Strand	Right	2.3

Table 4.4 Secondary Structures useful in matching onto polypeptides structures. Note 1 and 2 are used for competitive template generation. [12]

```

# Store the best and overall best after testing for it.
1  Best(c, ct)
2  partial_solution c
3  int ct
4  {
5    if (OverallBest.f < template[ct].f)
6    {
7      OverallBest  $\leftarrow$  (Overlay c  $\rightarrow$  template[ct])
8      Best[ct]  $\leftarrow$  (Overlay c  $\rightarrow$  template[ct])
9    }
10   else if (Best[ct].f < template[ct].f)
11   {
12     Best[ct]  $\leftarrow$  (Overlay c  $\rightarrow$  template[ct])
13   }
14 }
```

Table 4.5 Pseudo code for finding and storing the best fitness.

#	Store the best and overall best after testing for it.
1	<i>RotateBestIn()</i>
2	{
3	if (Panmetic)
4	{
5	worst = MinDBL
6	k = 0
7	for(<i>j</i> = 10; <i>j</i> < <i>ct(max)</i> ; <i>j</i> ++)
8	{
9	if (<i>Best[j].f</i> > <i>worst</i>)
10	{
11	worst = <i>Best[j].f</i>
12	k = <i>j</i>
13	}
14	}
15	for(<i>j</i> = 10; <i>j</i> < <i>ct(max)</i> ; <i>j</i> ++)
16	{
17	if (<i>Best[j].f</i> > <i>worst</i>)
18	{
19	template[<i>j</i>] ← Panmetic
20	else
21	{
22	template[<i>j</i>] ← <i>Best[j]</i>
23	}
24	}
25	else if (<i>Best[ct].f</i> < <i>template[ct].f</i>)
26	{
27	for(<i>j</i> = 10; <i>j</i> < <i>ct(max)</i> ; <i>j</i> ++)
28	{
29	template[<i>j</i>] ← <i>Best[j]</i>
30	}
31	}
32	}

Table 4.6 Pseudo code for rotating competitive template mechanism.

4.2.2 Farming Model. The farming model code modification is motivated by having the need for 2 parallel models working as one. The farming model is a simple dynamic load balancing implementation of the juxtaposition phase's evaluation function. See Section 3.7 for a complete description. Upon start-up, the fmGA initializes a pool of processors that is used in parallel to evaluate the fitness function of all new partial population members during phase of the algorithm. The only stipulation is that the total number of compute nodes must divide evenly by the number of algorithm nodes. For example, in the Figure 4.1 each Algorithm node is represented by a square and circles represent compute nodes. The configuration on the left is showing a configuration of one Algorithm node (which runs the fmGA) and three compute nodes. When this Algorithm node reaches the cut and splice phase within the juxtaposition phase, it builds the new population and then dynamically (according to the currently population size) divides up the evaluations between the three compute nodes - all left over evaluations are performed by the Algorithm Node. The configuration on the right side of Figure 4.1 shows how the communication occurs between Algorithm nodes plus from Algorithm node to compute nodes. This communication is essential when the Algorithm nodes are working together from a common population. In addition, there is another stipulation that no two Algorithm nodes can have a common compute node.

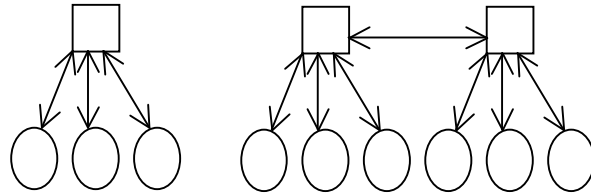


Figure 4.1 Farming model visualization of communication between algorithm nodes and farm or compute nodes. Algorithm nodes are represented by the square boxes and Compute nodes are the Ovals.

The farming model, built in code using message passing interface (MPI), follows the visual representation discussed above. The most important part is initializing the groups correctly. Once this is complete, all following MPI calls are the same

```

#
1  Initialize_Farms()
2  {
3  NumNodes = MPIcomm size command
4  NumFarms = Specified in Option File (Table 5.3)
5  NumFarmsPerAlg = Floor(NumFarms/NumAlg)
6  for ( $k = 0; k < NumFarmsPerAlg; k++$ )
7  {
8      alg_proc[k] =  $k * (NumFarmsPerAlg + 1)$ 
9      if ( $k * (NumFarmsPerAlg + 1) == \text{this\_node's\_world\_number}$ )
10     {
11         my_group = k
12         IamBoss = TRUE
13     }
14     for ( $j = 0; j < NumFarmsPerAlg + 1; j++$ )
15     {
16         farm_proc[k].ranks[j] =  $k * (NumFarmsPerAlg + 1) + j$ 
17         if ( $(k * (NumFarmPerAlg + 1) + j) == \text{this\_node's\_world\_number}$ ) &&  $j > 0$ )
18         {
19             my_group = k
20             IamFarm = TRUE
21         }
22     }
23 }

```

Table 4.7 Pseudo code for evaluation of partial solutions.

with one modification to the “group” identifier. Normally, all sends and receives are channelled to the entire group, MPIWORLD. Now, all calls are either to the a configured Algorithm Group or Farm Group. Furthermore, each farm group can only communicate with one algorithm node. The pseudo code for initialization is found in Table 4.7 and Figure 4.2 illustrates how the nodes are grouped. Communication only occur within a group. It is this restriction that forces the group relationships defined for this model. Further, it is easy to see that Algorithm nodes communicate only with Algorithm nodes and Farm nodes within that Algorithm node’s farm group. Furthermore, Farm nodes can only communicate with nodes that are within its own farm group – this always includes one Algorithm node.

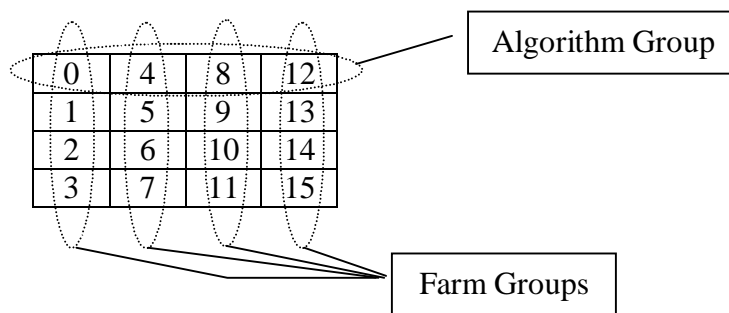


Figure 4.2 Visualization of nodes grouped into Algorithm and Farm arrays.

4.2.3 Building Block Analysis. No special implementation or code modifications were necessary for running these experiments. The parameters varied in this experiment were population size and building block size. The population size needed to be changed throughout the experimentation because as the building block sizes increased, so did the population size. Moreover, some of the building block sizes caused the population to grow very large which in turn causes system degradation. Therefore, the `n_a` value is lowered as the building block size rises to keep the population size to a reasonable number. This population size adjustment allows the algorithm to complete within a shorter length of time. The range of `n_a` values used are [10,50].

4.2.4 Protein 3D File Generation. The Protein Data Bank (PDB) format is well known in the biochemistry world. For purposes of generating protein conformations that can be viewed post-mortem, code was inserted to generate a PDB file after each building block size test. These PDB files contain cartesian coordinates of each atom which can be viewed with 3D molecular modelling software like Visual Molecular Dynamics (VMD). The implementation of code is rather simple; however, there is a need for precise placement of this code. Directly following an evaluation of a particular conformation, the structures needed in generating the PDB have the corresponding coordinates associated with that conformation evaluation. It is at

that time the PDB must be generated. In other words, the best conformation must be reevaluated just before generating the PDB for that particular conformation.

4.2.5 Multiple Objective. Code modifications were essential and extreme for the integration of the multiple objective approach. Furthermore, much time went in to the design of a modular Software Engineering approach to this implementation. Additionally, it was fortunate that there was existing code that striped pareto front points from formatted results. This same code was used in Multiobjective messy GA-II (MOMGA-II) [104]. By using existing code, focus was placed on the design of the fmGAMO and not on pareto front software.

The MOfmGA executes using the same basic algorithm structure as the fmGA. The differences are slight. First, the MOfmGA automatically uses a multiple competitive template design where each objective function is *assigned* a competitive template. This competitive template grows to optimize that particular objective function and each population member is overlayed onto this competitive template before evaluation. Secondly, as the Juxtapositional Phase completes population members (after overlaying onto a competitive template if necessary) they are written to a file for post mortem processing for pareto front points. Finally, after saving each Best as the next competitive template, the best found chromosomes are written out into a PDB file for viewing of the structure after completion or during execution of the program (Discussed in Section 4.2.4).

The general flow of the program is easy to follow; however, once the multiobjective part of the algorithm is applied, the solution determination is a little more complex. For this part of the code, on top of the parameters already discussed, we have to add in decision variables and objective functions. In addition to these variables, an implementation of multiple competitive templates in both separate and panmictic approaches are added as variables. The additional parameters are defined as follows:

Decision Variables: These variables are dependant upon the number of dihedral angles needed to define a protein. Worst case, this would be the $n-3$ dihedral angles where n is the number of atoms making up a protein. Fortunately, many dihedral angles are known to biochemists. Therefore, these angles can be set to the known angle and that particular dihedral angle can be removed from the decision variable parameter list. Ultimately, these decision variables reference back to the original angles defined within each residue thereby allowing calculation of the Objective Functions.

Objective Functions: Objective Functions for this particular design are held to be any combination of summations found in the fitness function. Moreover, the CHARMM model is built from numerous summations of energies found from designated configurations of atoms. The MOP design decomposes the CHARMM model into five different and separate objective functions allowing the researcher to select *any* combination of these functions – including a single objective approach using all five functions. Selection is set in a configuration file parsed by the algorithm before running a set of experiments.

[CHARMM] The energy function (Equation 2.7) as a whole sums the bond, angle, torsion, improper dihedral, water-water and van-der-waals potential energy between every atom. Moreover, all energy for both bonded and non-bonded forces are added. However, the MOP changes the sum by decomposing the energy function into five separate objective functions plus a fixed energy value. The fixed energy value is due to the assumed rigid structure of the protein of the *fixed* model chosen (bond length and bond angles between every three atoms are fixed to optimal). This fixed energy is to be accounted for within one of the objectives. In this investigation, the fixed energy is included in Objective 1. Within the MOGA code there are *eight* objectives that can be selectively added together to form any combination of *five* objectives. The nine functions are listed as follows:

1. Fixed Energy
2. Non-Bonded Energy
3. Non-Bonded Energy One-Four
4. Dependent Bond Energy
5. Independent Bond Energy
6. Dependent Angle Energy
7. Independent Angle Energy
8. Dependent Dihedral Energy
9. Independent Dihedral Energy
10. Independent Improper Dihedral Energy

Notice that there are ten functions listed. This is because the CHARMM energy function provides nine functions; however, the last function, *improper dihedral energy*, is not being utilized in these experiments because it is known to be insignificant compared to the cost of evaluation. The following five objectives were built from the nine available objectives listed above. Section of function is based on one criteria - type of bond. If the function is non-bonded energy related, it is added to objective 1. Following this, if the function is bonded energy related, it is added to objective 2. The fixed energy is always added to objective 1, as discussed earlier. The separation of the bonded and non-bonded energy function is not without reason. This selection allows the algorithm to optimize two meaningful natures: Physics and Chemistry. The Physics of the PSP problem focuses on keeping correct the coulomb interactions and steric anatomy of the protein. Whereas, the Chemistry of the PSP problem keeps correct the classical description of Chemistry. Moreover, dihedral angle, bond lengths, and bond angle energies are the focus of this objective.

- *Objective₁*

- ONE: Fixed and Non-Bonded Energy
- TWO: Non-Bonded Energy One-Four Energy
- *Objective₂*
 - THREE: Dependent Bond Energy and Independent Bond Energy
 - FOUR: Dependent Angle Energy and Independent Angle Energy
 - FIVE: Dependent Dihedral Energy and Independent Dihedral Energy

Essential modifications to the fitness function was the first to be changed. Calls to evaluate partial solutions had to incorporate the objective to evaluate. Objective declarations are accepted in the file used for defining multiple competitive templates. This is illustrated in Table 4.2. In addition to having these options, the researcher can also assign these 9 objectives to any of the 5 optional objectives within the code.

Additionally, each chromosome needs to have fitness values associated with each objective *and* competitive template. This means that the MOfmGA can handle both the multiple competitive templates and multiple objective functions and population member structures need to be modified for such type of experiments. Additionally, changes to storing the best/Overall best per Objective, evaluation per objective and tournament selection all needed to be made in support of the multiple objective code.

Modifications to evaluation calls were similar to the modifications made when adding multiple competitive templates; however, now the objective number must be traced. With few exceptions the code segment found in Table 4.8 wrappers each evaluation call within the single objective code.

This piece of code implies that for every index referring to a competitive template, it must be replaced by the "objective_number*ct + objective_number." This is true. In addition to this change, tournament selection must also add a section of code similar to what is found in Table 4.3 to account for all objective competitive


```

for ( $r = 0; r < num\_of\_objectives; r++$ )
{
  ## Original Evaluation Call (where  $ct = objective\_number * ct + objective\_number$ )
}

```

Table 4.8 Pseudo code modifying the evaluation function when moving from the single to multiple objective code.

4	$wins_0 = 0; wins_2 = 0$
4.1	for($i = 0; i < obj(max); i++$)
4.2	{
5	for($k = 0; k < ct(max); k++$)
6	{
6.1	$j = i * k + i$
7	if ($c_1.f[j] < c_2.f[j]$)
8	{
9	$wins_1 = wins_1 + 1$
10	}
11	else if ($c_2.f[j] < c_1.f[j]$)
12	{
13	$wins_2 = wins_2 + 1$
14	}
15	# add nothing if it is a tie
16	}
16.1	}

Table 4.9 Pseudo code for multiobjective tournament selection.

template evaluations. This is illustrated by replacing lines 4~16 in Table 4.3 with lines 4~16.1 in Table 4.9.

Finally, a mechanism was built to generate points representing population members at the end of the juxtapositional phase. The file containing the points representing the population are parsed for pareto dominant points at the end of the search. The process of finding pareto dominate points is defined in Definition 4 in Appendix K. The idea would be to get points that are on “Pareto Optimal” front (Definition 5 in Appendix K) [97]. The points are generated with the associated dihedral angles for backwards conformation visualization characterization; however, the objective functions are used for pareto dominance selection.

#	Technique
1	A separate plot for each model in the ensemble
2	Separate plots for just the Gly and Pro residues
3	Separate plots for each of the 20 different amino acid types
4	Separate plots for each residue in the sequence

Table 4.10 List of Ramachandran plot variations.

4.2.6 Ramachandran. Ramachandran plots may be generated by the four techniques found in Table 4.10. A good discussion of Ramachandran plot generation and usage can be found in [29].

Of these four Ramachandran techniques listed in Table 4.10, this investigation uses technique number 3 for implementation. The design of Ramachandran angle restriction is an involved process. The restriction of angles for each residue type involved creating a new C structure carrying information about each dihedral angle's type of angle and residue. Upon a call to compute a dihedral angle in radians (RAD), this structure maps that dihedral angle having a defined angle and residue type to a predefined range. There are three levels of ranges: Normal, Optimistic and Pessimistic. Worksheets establishing each range can be found in Appendix C. Illustrated in Figure 4.3 are a few mappings that might occur for an optimistical dihedral angle computation.

4.2.7 RMS difference. The last mechanism added to this investigation was the root mean squared difference calculations. Given an accepted solution to a particular proteins final folded state, this calculation determines the distance, in angstroms (\AA) $10^{-10}m$, a solution is from the accepted solution. There are two valid ways of calculating the RMS difference between proteins: 1) dihedral angle difference and 2) Cartesian coordinate difference. Both are implemented in this investigation. Equation 4.2 is used to calculate the RMS difference of the independent dihedral angles where $D_{\theta(i)}$ is dihedral number i within the found conformation and $D_{true_{\theta(i)}}$ is the dihedral angle number i within the accepted true conformation. Moreover,

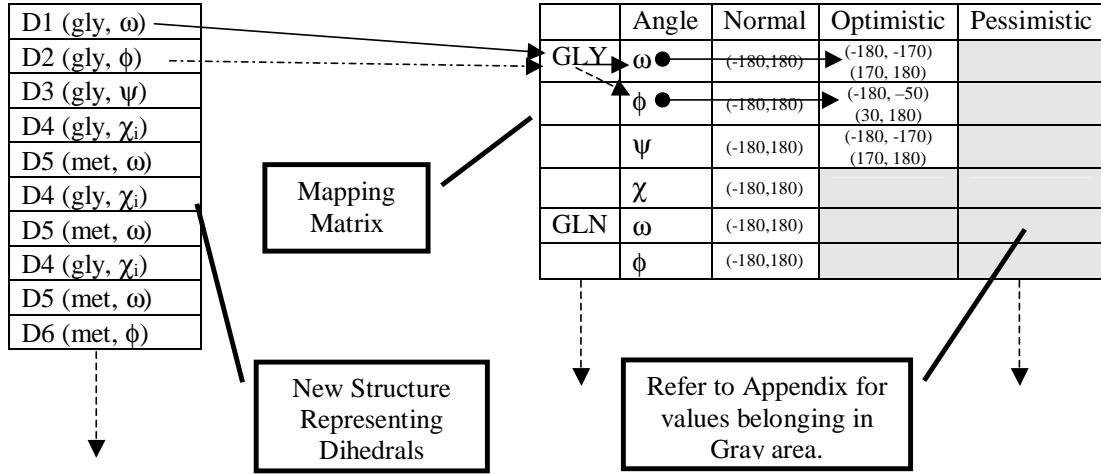


Figure 4.3 An example of how the mapping of the Ramachandran plots works in with the algorithm. For each evaluation, the mapping must be accomplished for each dihedral interpretation. This is quite computationally expensive.

Equation 4.1 is used to calculate the RMS difference of two structures where $A_{s(i)}$ is atom i from the test solution and $A_{t(i)}$ is atom i from the accepted solution.

$$[32] \sqrt{\sum_{i=1}^n (A_{s(i)}.x - A_{t(i)}.x)^2 + (A_{s(i)}.y - A_{t(i)}.y)^2 + (A_{s(i)}.z - A_{t(i)}.z)^2} \quad (4.1)$$

$$[74] \sqrt{\sum_{i=1}^n (D_{\theta(i)} - D_{true\theta(i)})^2} \quad (4.2)$$

4.3 Software Engineering Approaches

The basic software engineering practices are enforced throughout the software development. Academia produces much software that has little emphasis on maintainability, so was true with fmGA software previously used for solving the PSP problem. In attempt to forge better software, the current software was studied intensively before modifications were made. Upon recognizing the needed modifica-

tions c structs were built and an entire revamping of the current software began. The Specification Phase was obscure because there was only I who was both the system builder and client – thus, no formal specification documents was drawn. However, a data flow diagram was rendered for ideal understanding of current software execution and desired new algorithm flow. This design phase is illustrated by the Flow Diagram in Figure 3.11. The Project planning phase was rather short because of time constraints. The thrust of the effort was to make the program more modular and be able to more easily add proteins to the search. These goals were met. Finally, the testing for the program was accomplished and found to be both more efficient and effective than the previous software.

4.4 Summary

This chapter discussed the design methodology of code for experiments conducted in this investigation. The Multiple competitive template design is covered first, followed by integrations of the farming model and PDB file generation. Next is discussed the feature design for the MOfmGA and finally, design of the Ramachandran constraints and RMS difference calculation integration is covered. A few examples of mapping and pseudo code were given to show the design complexity and allow for reproducibility. Chapter 5 discusses the design of experiments and presentation and statistical methods used; furthermore, justification for experimental design, definitions the system and components under test, performance evaluation terminology, techniques and metrics are clarified.

V. Design of Experiments

Algorithm effectiveness and efficiency are the key criteria defining generic performance analysis of GAs. Effectiveness is quite an elusive metric for the PSP problem, but efficiency is not. This chapter discusses the study of both effectiveness and efficiency for the MOfmGA when used to solve the PSP problem. It presents justification of experimental design, system and components under test, system services, performance metrics, system and workload parameters, algorithm factors, justification for evaluation techniques, workload selection, hypothesis and presentation techniques.

5.1 Justification of experimental design

The experiments described in this chapter are selected to test both effectiveness and efficiency of the algorithm. A multiple objective experiment is the focal point of the thesis investigation because this experiment brings a new technique, meaning and validation to solving the PSP problem. Specific experiments testing effectiveness are the following: competitive template generation experiment, building block size experiment, Ramachandran constraint experiment and multiobjective experiment. Efficiency tested using the Farming Model experiment. Finally, extra mechanism to help judge the “goodness” of solutions found are the following: RMS difference and PDB file generation.

5.1.0.1 Competitive template experiment. These selected experiments are innovative techniques targeting the improvement of the fmGA. The fmGA explicitly manipulates building blocks (BB) in search of the global optimum and uses the idea of speciation through successive phases of the algorithm. The fmGA uses a competitive template, which is a fully specified population member, to evaluate these partially defined strings or building blocks. By focusing on modifying the process that the fmGA uses to create and update the competitive template during the

execution of the algorithm the algorithm’s effectiveness is increased. Therefore, the competitive template generation experiment is a good choice to test this hypothesis.

5.1.0.2 Building Block experiment. The building block (BB) analysis is performed in an attempt to identify the building block sizes that result in finding better solutions for POLY. A BB is a partial string representing bits from one, some, or all of the dihedral angles that each chromosome represents. The BBs are not restricted to be contiguous bits from the chromosomes but instead can be non-contiguous bits from the chromosome. Therefore if one purely looks at just one BB it may represent a whole dihedral angle or just various bits of multiple angles.

The BB analysis conducted covers a variety of BB sizes and compares the results to determine which size produces the best statistical results. One expects a BB size of 35 bits to yield the best due to the alpha helix [12] structure of POLY. Alpha helix proteins are known *a priori* to have 3.5 residues per turn [8].

5.1.0.3 Ramachandran experiment. Search algorithms having constraints on search space by a feasibility function statistically, overtime, must find better solutions. Moreover, solutions where fitness is known to be bad are removed. This premise also applies to this experiment, by constraining the search space to have only feasible solutions it is expected that solutions found must be better.

5.1.0.4 Multiojective experiment. In the single objective implementation of the fmGA, the CHARMM energy function is utilized and consists of a summation of several major terms. To utilize a multiobjective approach, the objectives are drawn from each of the terms within the CHARMM energy function. Specifically, the energy function is broken down into the connected (bonded) and non-connected (non-bonded) atom energies. These objectives are selected to decompose the problem into two goals to optimize: the Chemistry and Physics. It is

expected that by optimizing for each of these goals separately results found are more meaningful and overall better.

5.1.0.5 Farming Model. Alternate efficiency models, like the island model, have been used in the past; however these other models do not confront solely the computation time related to the fitness function. It is due to the complexities associated with the energy fitness function calculation and the fact that the fitness calculation is in the critical path of execution, the addition of a farming model is proposed. Farming out the fitness calculations to another set of slave processors allows for a decrease in the overall processing time as long as the computation time is greater than the communications time required. As the slave processors calculate fitness values the masters can do the same or conduct other computations. This experiment is expected to result in better overall efficiency of the algorithm run time.

5.2 System Under Test

The system under test (SUT) is AFIT's fmGA. The fmGA is programmed to run in serial and parallel on the following AFIT computer systems: Pile of PCs (PPCs), Cluster of Workstations (COWs) and Networks of Workstations (NOWs). The clusters of computers used in this investigation are defined in Appendix D. Computational requirements for simulation of a protein can be found in Section 3.3 and no computer system is available to perform to this requirement. It is for this reason that we use a stochastic search like the fmGA.

5.3 Component Under Study

When evaluating system performance it is easier to decompose the metrics into components so each component is isolated for a more accurate analysis. When measuring components, a unit of measure must be defined. Effectiveness and efficiency

are isolated performance metrics employed in this investigation. They are defined as:

5.3.1 Effectiveness Components. The Components Under Test (CUS) for effectiveness are the fmGA’s competitive template generation method, building block sizes at cut-off generations, and the number of objective used within the fmGA. Each component studies a different segment of the algorithm. The fmGA utilizes the competitive template throughout the algorithm; however, competitive template generation is only engaged within the start-up of the algorithm. Building block cutoff sizes are exclusively used during the Building Block Filtering Phase as *the* block size to reduce all population members before moving forward in the algorithm. Minimizing multiple objectives aims to solve the PSP problem by separating the problem into more meaningful partitions and solving for each of these partitions. Having multiple objectives changes the dynamics of the fmGA where the population becomes more diverse because the algorithm now keeps fit chromosomes for multiple objectives. Each one of these mechanisms is judged by how “good” the fitness of the overall best chromosome found. Fitness is calculated by the CHARMM energy model and results in units of kcal/mol.

5.3.2 Efficiency Component. The Component Under Test (CUT) for efficiency is how farming out the fitness function calculation makes good use of complimentary compute nodes. The farming out of a computationally expensive fitness evaluation should realize speed up in efficiency without affecting the effectiveness [35]. Wall clock time is measure in system clock time to complete in units of seconds.

5.4 System Services

The fmGA searches a constrained solution space, as discussed in Section 2.2.2, for lower energy values achieved by the protein at different conformations. Finding

the optimum fitness is not a realistic goal because the fmGA is a stochastic search algorithm; therefore, semi-optimal solutions are sought. The system allows a researcher to select from among numerous options. This includes but is not limited to the test protein, the number of epochs, building block size, and competitive template. The entire list of options can be found in Tables 5.1, 5.3, 5.4 and 4.2. It should be noted that the software can be written to be responsive to other variables as well; however, for each variable added to the options file, less memory is available for the algorithm itself.

As one can see by the number of options in Tables 5.1, 5.3 and 5.4, it would be difficult, if not impossible, to conduct a full factorial number of experiments with all the different settings. Options listed in **bold** are varied in the experiments. In addition, a new configuration file has been added extending these options further. This is illustrated in Table 4.2. Although the new options in Table 4.2 are not difficult to understand, Appendix E has been included to explain this file more thoroughly. The new options file makes changing objective and competitive template variables easier than if these options had been added directly into the original options file.

The number of possible outcomes for each experiment is the same size as the solution space (described in Section 2.2.2). It is known that the fmGA attempts to search the energy landscape in pursuit of finding a deep valley where a better conformation may be; however, there is no guarantee that it succeeds in finding a deep valley every experiment. Furthermore, it is known that the fmGA is a stochastic algorithm and searches with probability of error in finding the optimal solution. Every run yields similar, but different results. The fmGA completes each time when it finishes the number defined, by the research, generations and experiments.

5.5 *Design Discussion*

The design of the experiments is simply explained. Each experiment varied only one parameter. All other variables are kept the same for each related experiment.

Label in Configuration File	=	Description of setting and acceptable values
Random Seed	=	Seed setting for computer at start of experiment
Experiments	=	number of experiments to run
String Length	=	Number of bits representing a complete chromosome
Block Size (min max)	=	The Range of Block sizes to use for experiments
Genetic Alphabet	=	'01' is specified for a binary alphabet
Encoding	=	0=Binary code 1=Gray
Shuffle Number	=	Number of times population is mixed before selection
Cut Probability	=	($0 \leq \text{Number} \leq 1$):Probability of cutting a member
Splice Probability	=	($0 \leq \text{Number} \leq 1$):Probability of splicing a member
Overflow (> 1.0)	=	Coefficient of string growth ^a
Sweeps	=	Number of sweeps each CT gets after creation ^b
Competitive Template Guesses	=	Number of tries to generate a good CT
Secondary Structure (SS) Fraction	=	Fraction of population is needed to identify SS pattern
Good Population Fraction	=	Enriches population until this good member ^c %
Pop Type 0-C 1-S 2-B 3-A	=	Type of population member to enrich the population ^d
Initial Energy Cutoff Value	=	Cut-off fitness value - Fitness eval stopping criteria
Protein Structured Used	=	Protein used in experiment (POLY or MET)
Number of Residues in Protein	=	Number of residues in Protein ^e
Heterogeneous	=	Yes/No if computer systems are Heterogeneous

^achromosome lengths are set by the string length during the Cut and Splice function string may grow larger than this number. The Overflow is selected to estimate how large the researcher thinks a string might reach.

^bThese are used during non-multiple CT experiments.

^cDuring the primordial phase the percentage of good population members is monitored and when the fraction of good population members divided by the population size falls below this given fraction, good population members are produced to replace bad ones until the percentage has returned to the specified size.

^dWhen good population members are being added in the Primordial Phase, the newly generated population members are added according to a particular type. (0=S or Randomly pick 1,2 or 3 for member Creation, 1=C or Normal Sweeping of bad member, 2=B or Beta Sheet member creation, 3=A or Alpha helix member creation).

^eThe number of residues indicated in the option file is different than the total number of variable dihedral angles

Table 5.1 Options for the fmGA

Experiment	Parameters
*Competitive Template	generation method and number of competitive templates
Building Block	building block cut-off sizes
Farming Model	algorithm nodes and compute nodes
Multiobjective	generation method and number of competitive templates
Feature experiment	Single vs. Multiobjective (comparison of experiments noted with *)
Ramachandran	variety of constraints set on dihedral angle

Table 5.2 Parameters for design

Furthermore, each experiment is run 10 times for statistical purposes. Parameters used for each experiment are found in Table 5.2.

5.6 Performance Metric

The algorithm’s generic performance metric is two fold - time to converge (efficiency) and the *goodness* of the found energy level (effectiveness). The fmGA uses the CHARMM, version C22 (described in Section 2.5), model to calculate the energy level of a protein in a particular conformation. The CHARMM energy model is an example of a force field energy model measuring potential energy of the conformation. Energy is measured in kcal/mol units. With this fitness function, lower energy equals a better solution.

5.7 Parameters

All system and workload parameters are displayed in Table 5.5. In the following sections a description of both system and workloads is presented.

5.7.1 System Parameters. AFIT’s fmGA was developed by previous AFIT masters students [32][68] specifically to solve the PSP problem. These students conducted numerous experiments using this particular fmGA providing baseline data from which to compare current results. Previous test revealed advantageous population sizes, number of sweeps [74] and number of generations to employ [42]. In addition to these parameters, doping the population with a percentage of good pop-

Label in Configuration File	=	Description of setting and acceptable values
Ramachandran Plot	=	Specifies if Ramachandran constraints are used ^a
Population Recombination	=	Specifies type of recombination method to use
Initialization Flag	=	Specifies type of initial population members to create ^b
Distinct Competitive Templates	=	Specifies CT is supplied as first and best CT found
Competitive Template Supplied	=	Yes/No if CT is given for use experiment
Global Selection	=	Specifies if Global selection is applied (pfmga)
Migration	=	Specifies if Migration is applied (pfmga)
Verbose	=	Yes/No if Verbose mode is on or off (debugging)
Energy Farms	=	Number of slave nodes available to farm fitness evals
Panmestic	=	Panmestic CT implimented
Conjugate Gradient	=	(0 ≤ Number ≤ 1):Probability of applying CG
Baldwinian or Lamarchkian (B/L)	=	B=Baldwinian, L=Lamarchkian, Y=Both, N=Neither
Probablity B/L	=	(0 ≤ Number ≤ 1):Probability of applying Baldwinian
RMS Calculation	=	A=All atom diff, B=Backbone atom diff, N=Turn Off
Primordial Generations	=	200 200 200 200 200 ^c
Total Generations	=	400 400 400 400 400 ^d
N_a	=	50 50 50 50 50 ^e

^aRamachandran Plots can be N=Not Used, O=Optimistically applied, or P=Pessimistically applied. See Appendix C for defined optimistic and pessimistic Ramachandran values.

^bInitial population members may be either R=Random or C=Complimented.

^cSpecifies how many primordial generations are conducted before moving to the next Phase for each block size specified above in the block size option.

^dSpecifies the total number of generations used in each phase. Implying that the generations for the Juxtapositional Phase is equal to the total less the number of Primordial Phase generations.

^eSpecifies the n_a in the population sizing formula (Equation 3.8).

Table 5.3 Options for the fmGA

Cut-off generation ^a	String length ^b	Threshold ^c
0	220	200
13	180	140
39	148	128
45	125	105
51	109	99
72	94	83
96	26	20
97	25	19
98	24	18
99	23	17
103	22	16
105	21	15
107	20 ^d	14
112	19 ^d	13
117	18 ^d	12
122	17 ^d	11
127	16 ^d	10

^aThe cut-off generation is the generation at which BBF phase is conducted.

^bThis is the size of every string after the BBF phase is completed.

^cThe threshold is how far apart the BBF phase enforces a Hamming Code distance between partial solutions.

^dThe last string lengths indicated must correspond to the building block size window chosen. For example, the Minimum and Maximum block sizes are 16 and 20 respectively – meaning that the min, max and all sizes in between must be specified in the last rows of the string-length column as they are in this example.

Table 5.4 Input Schedule for the fmGA

Selected Parameters	
System	Workload
fmGA	MET
Serial	POLY
Parallel	
fmMOGA	
Building Block Size (13 levels)	
Competitive Template Generations	
Alpha Helix	
Beta-Sheet	
Random	
Multiple types	
Panmetic	
No Sweeps	
Multi Objective	
Two Objectives	
Ramachandran Plots (3 levels)	

Table 5.5 System and Workload Parameters

ulation members had also been studied [74]. This work is followed by the multiple competitive template, building block size, farming model, Ramachandran and multiobjective experiments. Comparisons can be drawn between these previous studies and this thesis effort. System parameters for this investigation include the fmGA, building block sizes, competitive template generation techniques, number of objectives and Ramachandran plots. Additionally, drawing on past results of parameter setting, the number of sweeps and generational schedule (discussed in Section 3.7) is set to optimal.

5.7.2 Workload Parameters. Two proteins are specifically chosen as the workload (POLY and MET). These define the atom organization that the fmGA uses as it searches for a good fitness. We pick two proteins with entirely different geometric conformations for a comparison. Both are relatively small proteins, but the POLY protein is a bit larger - having 14 amino acids compared to MET's 5. Evaluating this larger protein should allow us to speculate on how long it might

take the fmGA to converge on an answer for even larger proteins. Furthermore, POLY has an alpha helix secondary structure which allows us to apply *specialized* techniques in finding special secondary structures.

5.8 Algorithm Factors

Performance factors in this experiment are the workload parameters, competitive template variations, Multi Objectives, and new Ramachandran plots [26]. Changing the design method of our competitive template allows us to compare previous implementations of this fmGA with those proposed in this experiment. Additionally, having multiple proteins allows us to gather fitness on two entirely different protein structure. If we observe improvement using our new competitive template which uses domain information, we may also observe that the improvement might be correlated between the competitive template’s geometric configuration and the geometric configuration of the protein under test.

Where k is the number of factors, a full factorial experiment would require $n = \prod_{i=1}^k$ [47] experiments. The total number of experiments would be 2 (Protein) x 3 (Levels of Ramachandran plots) x 6 (Competitive Template Generation Method) x 2 (Type of Objective) x 13 (Building Block Sizes) x 2 (Parallel/Serial) equating to 1872 experiments. Additionally, 10 replications of each experiment must be performed totaling 18720 experiments. This was an extraordinary number of experiments to accomplish; therefore, each experiment was conducted and compared separately – only comparing the final best solutions. Moreover, experimental analysis for multiobjective plus competitive template generation was conducted separately from single objective plus competitive template generation. Parallel and serial experiments were conducted together for they targeted efficiency, not effectiveness. Ramachandran experimentation and building block sizes experiments are all conducted separately. The idea was to fine tune the fmGA in finding good solutions for these two proteins: POLY and MET.

5.9 Hypothesis

The hypothesis of the outcome of these experiments is that the multiple objective approach using multiple competitive templates acquires the best effectiveness results. Furthermore, the farming model is expected to show that speedup can be obtained by farming out the fitness function because it is the computational bottleneck. In addition to results of these experiments, it is expected that using the constraints of Ramachandran plots is advantageous to finding even better solutions. Furthermore, The accuracy of the tests should be sufficient to allow us to determine if the new implementation is better than previous versions.

5.10 Evaluation Techniques

Sufficient time is available to implement changes to the fmGA and run outlined experiments on both proteins (MET and POLY). Additionally, tools are in place to accomplish any code modifications required.

Techniques such as the RMS difference calculation and graphical visualization comparison allow one to identify and quantify the amount of differences between the actual and experimentally derived geometric structures. Proteins are known to fold into different shapes even though they are made of the same exact atoms and bonds. Therefore, it is nearly impossible for us to use our experimental conclusions supported by X-ray Crystallography to establish, without any doubt, that we have found the correct geometric conformation for a particular protein. Moreover, this effort is two fold in an attempt to find acceptable solutions to the PSP problem and evaluating the fmGA's ability to be adapted to hard problems like the PSP problem.

5.11 Workload Selection

Selecting the workload for the system is procedural in this experiment. Any protein selected would exercise all the services; however, if we choose a large protein it would have overloaded our SUT because of the combinatorics described in Section

Protein	Residues(Sequence)	2ndary Structs	~Atoms	Search Space
MET ^a	5 (Tyr-Gly-Gly-Phe-Met)	<i>none</i>	87	2^{1615}
POLY ^a	14 (Ala-Ala-...-Ala)	α - <i>helix</i>	182	2^{3420}
Tufstin	4 (Thr-Lys-Pro-Arg)	<i>none</i>	86	2^{1596}
BETA ^b	16 ((Ala - Glu) ₂ - (Ala - Lys) ₂) ₂	β - <i>sheet</i>	~176	2^{3325}
Cramblin	46 (See PDB 1AB1)	2- β - <i>sheets</i> , 2- α - <i>helix</i>	329	2^{6213}
Protein L	78 (See [59])	β - <i>strand</i> , α - <i>helix</i>	605	2^{11457}

^aInitiating the workload proteins. Future research may include others listed in table.

^bhas a characteristic β -sheet circular dichroism spectrum in water. Upon the addition of salt, the peptide spontaneously assembles to form a macroscopic membrane. [103]

Table 5.6 List of possible workload parameters along with their associated search space.

2.2.2. Some examples of proteins and their associated search space are listed in Table 5.6. In addition, the new system parameters needed to be exercised on not one, but two separate workloads. Therefore, two small proteins were chosen. A larger protein would take a much longer evaluation time; however, might add some more insight to solving the problem. Yet, for this investigation, these two small proteins are appropriate for timeliness and level of detail required. The selection of MET and POLY are sufficient to reach the thesis goals stated in Section 1.2.

5.12 Experimental Design

1. The first experimental design is a full factorial experiment of competitive template generation methods. Our first variant consists of changing our method of generating the competitive template. The second variant is a choice of proteins. This experiment is accomplished twice; however, the results of the second is provided in this investigation. The experiment's first tool selection is the original fmGA and the second tool is the MOfmGA. Accomplishing both these experiments gave insight to new code capabilities and correctness. This met Objectives 2 and 3 of goals outlined in Section 1.2
2. The second experimental design is a full factorial experiment of parallel/serial mode of the fmGA. The only variant is that of using parallel versus serial mode

and validating that the fitness does not change when evaluating on different computer nodes. This met Objective 2 of goals outlined in Section 1.2.

3. The third experiment design is a full factorial experiment of multi objective plus competitive template generation techniques. The first variant consists of changing the competitive template generations techniques in conjunction with splitting the objectives. These experiments can be compared to the single objective in experimental design one. This met Objective 3 of goals outlined in Section 1.2
4. The last experimental design is a full factorial experiment of Ramachandran plots. The first variant is that of the three levels of the Ramachandran plots (Optimistic, Pessimistic and none). Details of these values can be found in Appendix C. The second variant is the protein selection for evaluated. This met Objective 4 of goals outlined in Section 1.2

Accordingly, each experiment is run 10 times. This number is suggested by [47] as a good number from which to be able to draw results. To estimate the experimental errors, we use the student-t distribution analysis, paired observations test and the Kruskal-Wallis test [32]. In identifying which method is considered as “better”, a difference of systems is used. This met Objective 6 of goals outlined in Section 1.2

5.13 Analyze and Interpret results

After gathering the data, a regression analysis is conducted to identify the relationship of the data (e.g. exponential, linear, logarithmic, etc). With the data model identified, we can compare previous data [74] to the new data by testing the differences between these samples and check for a zero mean using an unpaired samples test. This difference analysis might also provide us with better insight to how many observations should be collected if we want to be sure that any two implementations are different.

5.14 Statistical Techniques

5.14.1 Kruskal-Wallis. The Kruskal-Wallis H test is the main statistical method used in for the determination if two samples are from the same population. This test is primarily used when no knowledge of the type of distribution is known; however, it can be shown that the sampling distribution of H is nearly a chi-squared distribution with $k - 1$ degrees of freedom, given that N_1, N_2, \dots, N_k sum to at least 5 [93]. The definition of the Kruskal-Wallis H Test (H Test) is the following:

$$H = \frac{12}{N(N-1)} \sum_{i=1}^k \frac{R_i^2}{N_j} - 3(N+1) \quad (5.1)$$

- Given

k sample sizes $N_1, N_2, \dots, N_k \therefore N = \sum_{i=1}^k N_i$

k samples are all ranked together according to size \therefore the ranks are R_1, R_2, \dots, R_k

Upon calculation of H using Equation 5.1, this value, H , is treated as though it were a value of chi-square sampling distribution with the degrees of freedom(df) = k-1. This nonparametric method for analysis of variance for one-way classification, or one-factor experiments, and generalizations can be made [93].

5.14.2 t-test Paired/Unpaired Observations. A second statistical method for an analysis of variance is the Student t-test. This test can be applied to both paired and unpaired observations; however, the application for each is quite different *and* they have much different meaning as to differences. [47]

- Paired Conducting n experiments on two different algorithms such that there is a one-to-one correspondence between the i th test on algorithm A and the i th on algorithm B. Two samples are treated as on sample of n pairs. Each pair's difference in fitness found is then computed and a confidence interval is constructed for this found difference. Confidence intervals including zero represent

algorithms that are *not* different. This is useful if the number of experiments were one – no distribution. However, this method does not include the variance per experiment making it an unfit model for these types of experiments. Furthermore, this type of analysis help a researcher to determine if two paired experiments are different excluding the variance in *paired* samples.

- Unpaired This analysis requires for the observations to be unpaired in that there is no correspondence between the i^{th} test on algorithm A and the i^{th} on algorithm B. This test can be applied for each sample grouping of paired experiments giving the researcher an idea if the data is different that sample grouping from another experiment; however, this analysis assumes a gaussian distribution. Again, the inclusion of zero within the confidence interval results if the Algorithms are not different.

5.15 Presentation Techniques

Presentation of experiment data comes in a few different “flavors”. Generational plots are necessary to show how different techniques compare in a progressional search. These generational plots are building block test versus fitness plots. Also, scatter plots as well as bar charts are used to characterize the distribution of fitness values in the search landscape. Additionally, new fitness versus RMS Cartesian Coordinate and Dihedral angle difference plots are used to valid the CHARMM fitness function for only one protein. Finally, a 3D visualization of each protein is provided to give biochemists a graphical representation of a semi-optimal solution found by the MOfmGA. This meets Objective 7 of goals outlined in Section 1.2

5.16 Summary

This chapter discusses the justification for selected experiments used in a statistical attempt to show algorithm adaptations are advantageous to finding better protein conformations in a shorter wall clock time. The experiments, factors, met-

rics, SUT, CUT, parameters, statistical methods and data visualization techniques are described. Additionally, a hypothesis is drawn of expected outcomes for these experiments. Chapter VI analyzes the results found for each experiment. Presentation techniques described in Section 5.15 are used in the analysis.

VI. Results and Analysis

This Chapter focuses on experimental results and analysis as well as resolving the computational model to the biological model. Each of the experiments discussed in Chapters 4 and 5 are addressed along with statistical evaluation methods. Additionally, parameter selection and interpretation for each of the following experiments is provided: Multiple Competitive template, Farming Model, Building Block Size, Protein 3D File Generation, Multiobjective, Ramachandran and RMS dihedral angle and Cartesian coordinate differences.

6.1 Multiple Competitive Templates

The multiple competitive template experiment is our first design modification to the fmGA. As discussed in Section 4.2.1, this modification requires the fmGA to have the ability to compute a panmictic competitive template¹ in addition to having multiple competitive templates present during computational search.

6.1.1 Results for Multiple Competitive Template Experiment with MET.

6.1.1.1 Effectiveness. Generation of the Alpha-helix competitive template overall produced the worst results followed by the Randomly generated competitive template. The beta-sheet competitive template generation was next best, then the Panmictic competitive template and finally the best was the Alpha, Beta and Random competitive templates. This is illustrated in Figure 6.1. Please note the error bars reflecting the 85% confidence intervals from plotted values. Error bars are generating using Student t-test table, assuming normal distribution [47]. These error bars are useful to visually identify variance of values plotted. This test reveals that if these are normally distributed points these methods can be grouped into two groups as being of the same performance. The alpha-helix and randomly

¹A panmictic competitive template is derived from the existing multiple competitive templates.

competitive template generation method can be grouped as over the worst performers and the Beta-sheet, multiple and panmictic competitive template can be grouped as overall better performers. The observation that the Random and Alpha-helix competitive template generations techniques are the same is supported by the second statistical method studied, paired observations. Results of this test can be found in Figure 6.2. It concludes that the rest of the competitive template techniques are different. Finally, a Kruskal-Wallis test is conducted. Results of which support the first proclamation that each method is different and found to be in the performance order stated at the beginning of this paragraph. One final attribute is worth mentioning; although the randomly competitive template is statistically proven to perform worse than all but one other method, it did produced the lowest fitness, -34.11 kcal/mol, found in all of these experiments. The entire data value set can be found in Appendix L.

6.1.1.2 Efficiency. Figure 6.3 illustrates the time it took each competitive template approach to complete. The most time consuming approach was the multiple competitive template method – no doubt due to the extra number of fitness evaluations for each evaluation. All other methods require the same amount of time to complete.

6.1.1.3 Conclusion for Multiple Competitive Template Experiment with MET. According to Figures 6.1 and 6.3, the panmictic competitive template is the best balance between efficiency and effectiveness for a protein such as MET. However, if effectiveness is needed by the decision maker, then the both the multiple competitive template and randomly generated competitive template designs should also be used. The randomly generation method found the overall best fitness for MET in this experiment and it is statistically proven that the multiple competitive template method is better than all other methods.

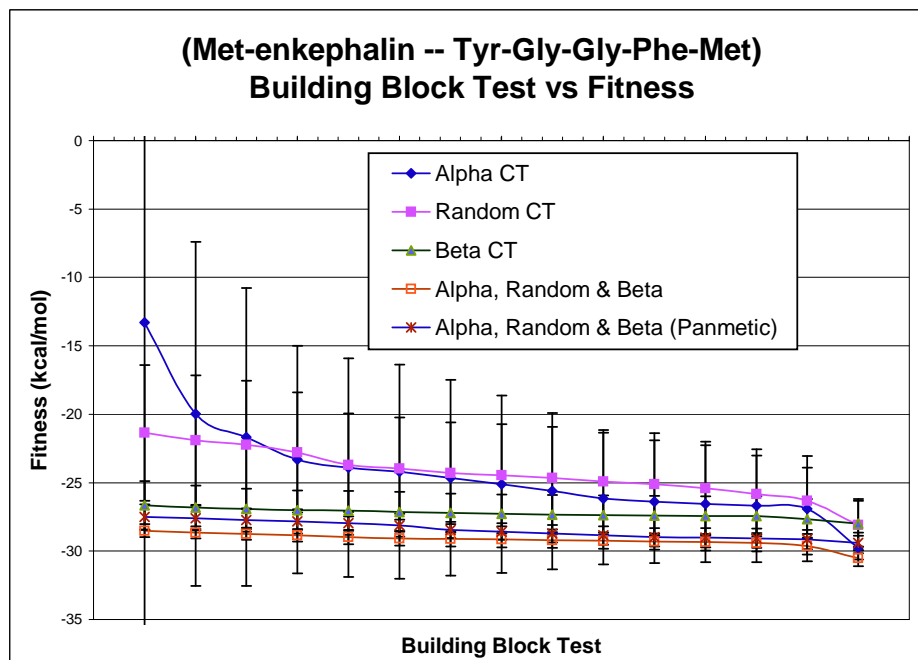


Figure 6.1 Building Block Test vs. Fitness plot of results for an experiment using multiple methods of competitive template generation on the protein MET.

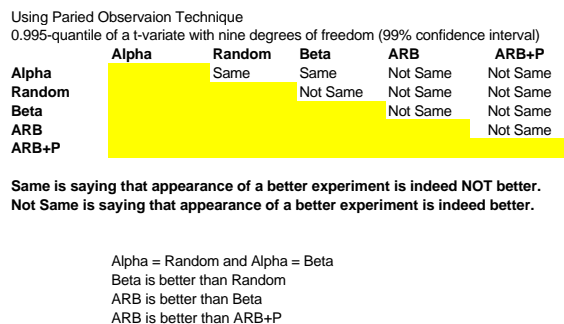


Figure 6.2 Summary of results after conducting the paired observation difference test found in [47] on the multiple competitive template experiment with MET.

6.1.2 Results for Multiple Competitive Template Experiment with POLY.

The results of the multiple competitive template experiment using POLY as the workload is more interesting from an algorithm to a biological point of view. We expect that the Alpha-helix competitive template generation techniques to outper-

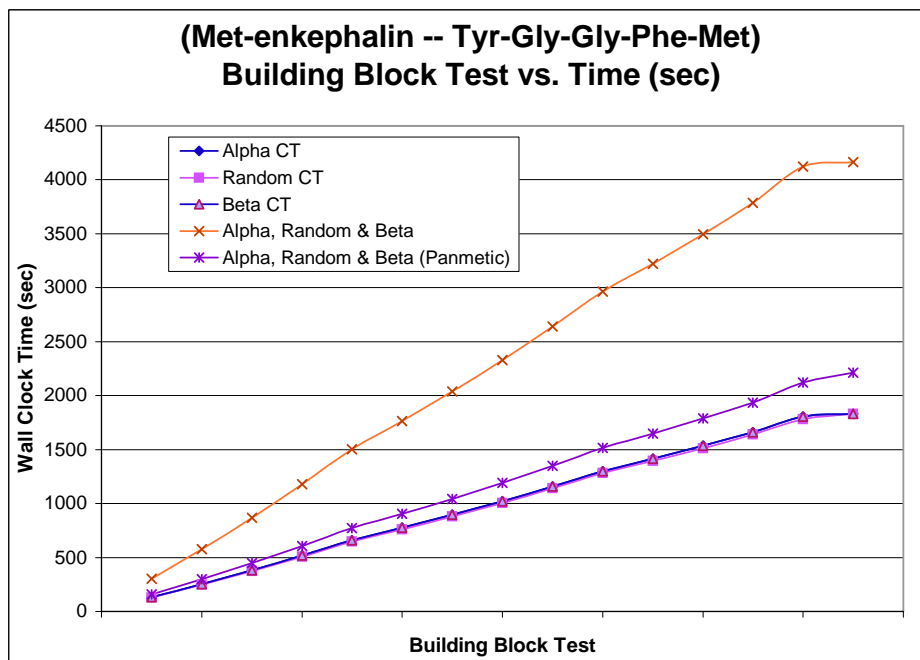


Figure 6.3 Building Block Test vs. Time to Complete plot of results for an experiment using multiple methods of competitive template generation on the protein MET.

form any technique without this. The reason for this hypothesis is that it is known *a priori* that POLY folds into an Alpha-helix conformation [12].

6.1.2.1 Effectiveness. Generation of the Alpha-helix produces good results; however, the multiple and panmantic competitive template methods also performed well – even better in fact. Both the Beta-sheet and randomly generated competitive template generation approaches performed the worst. This is illustrated in Figure 6.4. The Student t-test is used to draw the 85% variance bars in Figure 6.4 and it can be concluded with respect to this statistical method that there is a clear difference between the Random, Beta-sheet, and all Alpha-helix related templates (Alpha-helix, multiple, and panmantic competitive template). Similar results are reported with the paired observation statistical analysis. The Alpha-helix and multiple competitive template methods are reportedly the same, as all others are considered different. Accordingly, the paired observation test has concluded that the order from

best to worst is the following: panmetic, multiple, Alpha-helix, Beta-sheet, and randomly generated competitive template method. Finally, Kruskal-Wallis test also confirmed that the Alpha-helix related, Beta-sheet, and randomly generated competitive template methods are also different. It concluded conceptually that it was 84% confident that these are different (Chi-squared distribution with 2 degrees of freedom and 3.65 quantile) and computationally it is 100% confident that these are different (Chi-squared distribution with 2 degrees of freedom and 924 quantile). A further Kruskal-Wallis test is conducted on the three Alpha-helix related competitive template methods. Conceptually, it concluded that there was no difference between the three (94% confident using $df=2$ and 0.12 quantile of the Chi-squared distribution). Moreover, computationally Kruskal-Wallis test concluded that these are 45% confident that these are the same. The computational Kruskal-Wallis is more strict when it comes to concluding things are different; therefore, we can conclude after analyzing the data using three different methods that these three better competitive template methods are the same statistically speaking. Although the panmetic is found to be statistically the same as the other two Alpha-helix related competitive template methods it achieved the best fitness of -172.1 kcal/mol.

6.1.2.2 Efficiency. Again we see a similar occurrence with the increased time for the multiple competitive template approach while all others are rather similar in time. This is due to the number of extra calculations made when keeping three competitive templates. It is a good thing that the time usage is not proportional to the number of competitive templates in use.

6.1.2.3 Conclusion for Multiple Competitive Template Experiment with POLY. The panmetic competitive template is the best balance between efficiency and effectiveness for this protein as well as MET. Not only is it a good performer time-wise, it also found the overall best fitness for the entire experiment. Future experimentation should be conducted on different proteins having the same Alpha-

helix secondary structure to confirm the usefulness of this competitive template method.

6.2 Farming model experiment

The pfmGA utilizes an island model [32] paradigm to conduct parallel communications between processors (See Section 3.7 for pfmGA details). At each stage of communications, all of the processors communicate their best found population member to processor 0. Processor 0 then determines which is the “best” and communicates that population member back to all of the processors who then update their competitive template. After the update, all of the processors continue to execute the algorithm independently with independent population members until the next update communication is necessary.

Due to the complexities associated with the energy fitness function calculation, the addition of a farming model *in combination with the island model* is proposed. Farming out the fitness calculations to another set of slave processors allows for a decrease in the overall processing time as long as the computation time is greater than the communications time required. As the slave processors calculate fitness values the masters can do the same or conduct other computations. In addition to speedup gained for the workload proteins selected in this investigation, the addition of these slave processors allows for the MOfmGA to handle larger proteins.

With the addition of farms, the program becomes multiple program multiple data (MPMD). There is an advantage to having the ability to execute multiple parallel models. The advantage lies in the structure of the communications and memory of the computer platform of choice. In the single data model, the GAs execute in parallel and generate populations separately from one another, with interactions only occurring when a migration of a good population members occurs with some probability. This model is advantageous to use when the communication cost is high. However, if one has access to a shared memory machine, a Global population

model may be more appropriate since communication cost may not be as much of a concern. This shared memory setup depicts a MPMD where data does not need to be transferred among processors and data pipelining can be utilized more readily (requirement of control parallelism). The systems used in these experiments are not currently shared memory systems but the communication cost is later shown to be insignificant compared to the cost of the energy fitness function evaluations.

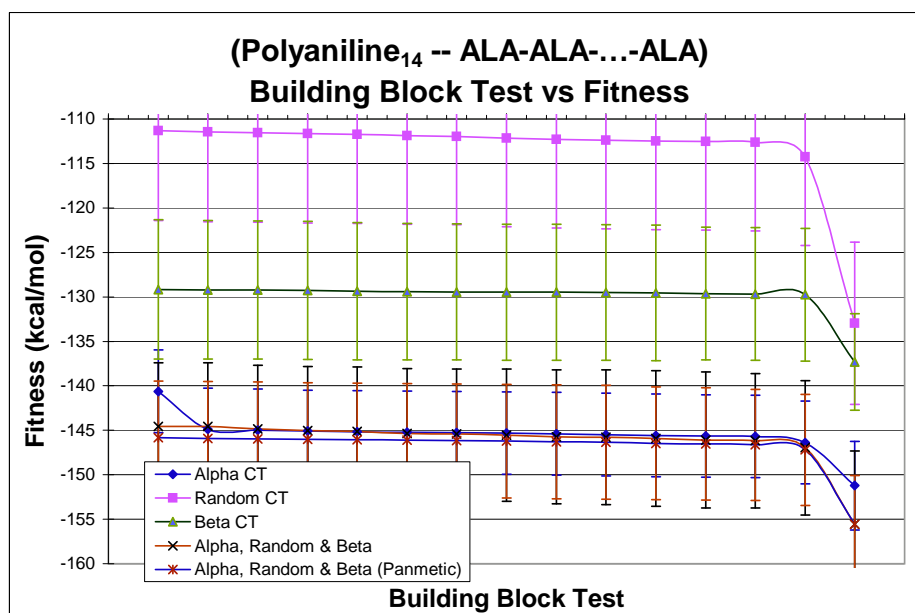


Figure 6.4 Building Block Test vs. Fitness plot of results for an experiment using multiple methods of competitive template generation on the protein POLY.

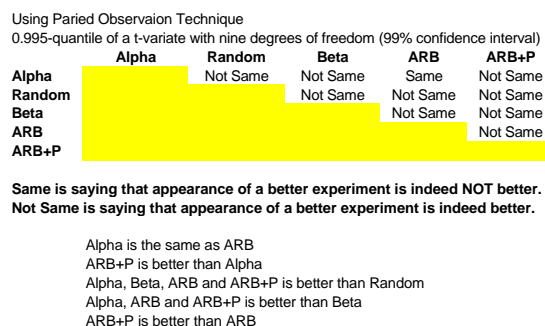


Figure 6.5 Summary of results after conducting the paired observation difference test found in [47] on the multiple competitive template experiment with POLY.

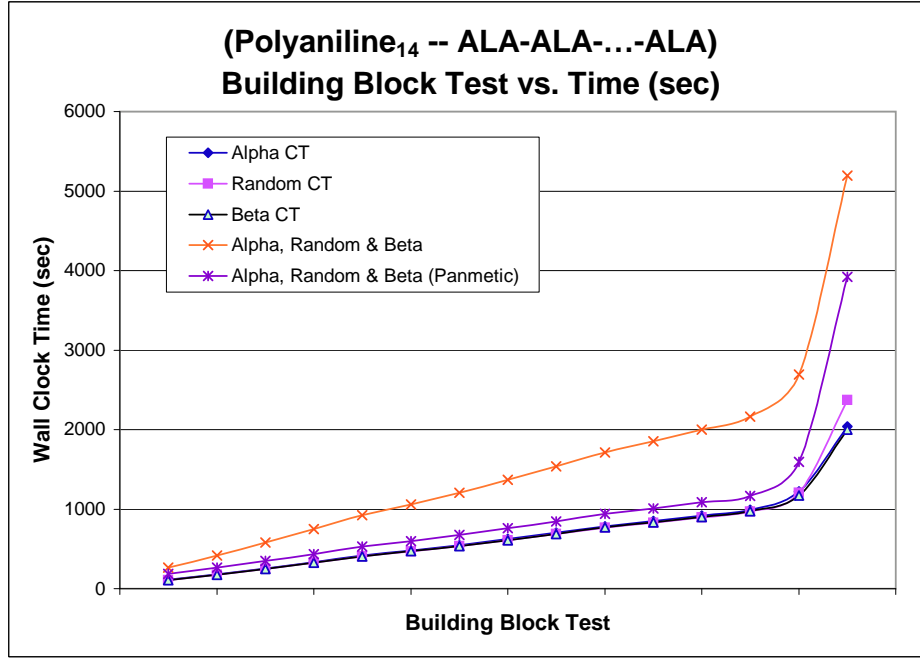


Figure 6.6 Building Block Test vs. Time to Complete plot of results for an experiment using multiple methods of competitive template generation on the protein POLY.

The following pfmGA parameters are kept constant (set at standard values) throughout all of the testing: string length = 560 bits, cut probability = 0.02, splice probability = 1.00, primordial generations = 200, juxtapositional generations = 100, total generations = 300. An input schedule is also used to specify during which generations BBF occurs. Computer systems used in this experiment can be found in Table M.1 in Appendix M.

6.2.1 Farming Experiment. The farming model is a dynamic load balancing implementation to increase the efficiency of the algorithm when additional processors are available. By definition, the fitness evaluations from the Juxtapositional Phase are farmed out to additional processors. Fitness function complexity can be found in Table 6.1 [32]. Upon start-up, the pfmGA initializes a pool of processors to be used for parallel evaluation of the fitness function of each population member. By design, the total number of processors used must be an evenly divided by the number of

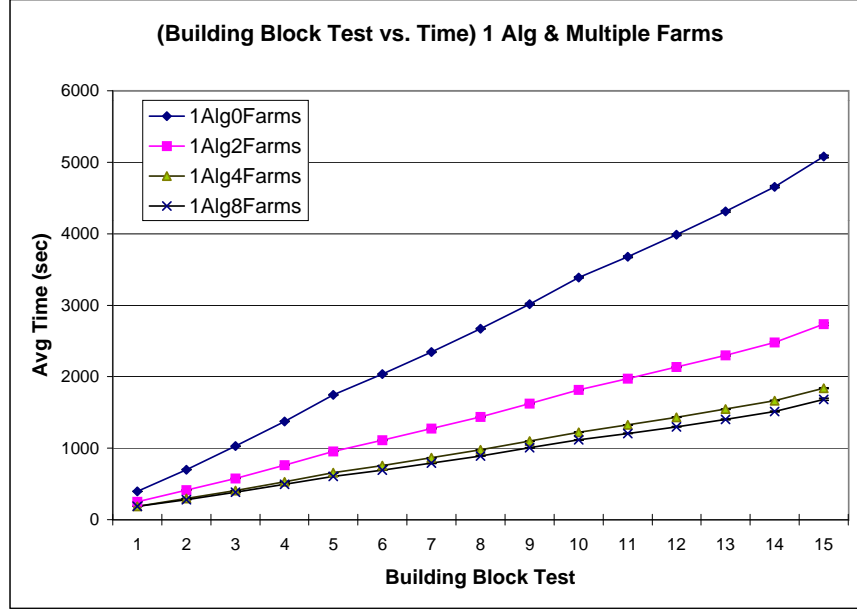


Figure 6.7 Time vs. BB Test for 1 Algorithm Node. Validation of fitness values before and after using the Farming model can be found in Appendix M

Table 6.1 Time Complexity of Energy Minimization Methods [32]

Energy Calculation Method	Time Complexity	Time Estimate for $n = 1000$
<i>ab initio</i>	$\mathcal{O}(n^5)$	11.5 days
<i>semi-empirical</i>	$\mathcal{O}(n^4) - \mathcal{O}(n^3)$	17 min - 1 sec
<i>force-field</i>	$\mathcal{O}(n^2)$	1 msec

Algorithm nodes. Figure 4.1 illustrates an example of a single Algorithm node and three farming nodes on the left and two Algorithm nodes with three farming nodes per Algorithm node on the right. This figure also illustrates that the Algorithm nodes have the ability to communicate with each other if necessary, but farming nodes are not shared across Algorithm nodes.

A goal of this testing is to determine the speedup associated with increasing the number of farming processors per Algorithm node in the pfmGA. Figure 6.7 illustrates a plot of one Algorithm node with a number of different farming nodes. Each Building Block (BB) test point represents the average value for a specific BB size (in increasing order) executed by the pfmGA. As the BB size increases, the

average execution time also increases as one would expect because the population size grows as the BB size grows. Additionally, one can see from Figure 6.7 that as the number of farming processors increases, the average execution time decreases for any given BB test. In this test there exists a significant improvement in modifying the number of farming nodes from 0 to 2 and from 2 to 4. An increase in the farming nodes from 4 to 8 provides some improvement but this improvement is relatively small. The reason for this is that as the number of farming nodes is increased for a specific population size, the amount of computational work that each node completes decreases. Eventually the communications time would become greater than the computation time per node and would yield additional farming nodes a detriment to the experiment. Still, the best speedup obtained was with 8 farming nodes where the serial time was 5080 seconds while the parallel time was 1684 seconds yielding a speedup of 3 times. This validate our model and we can draw a conclusion that this model increases the efficiency of the fmGA.

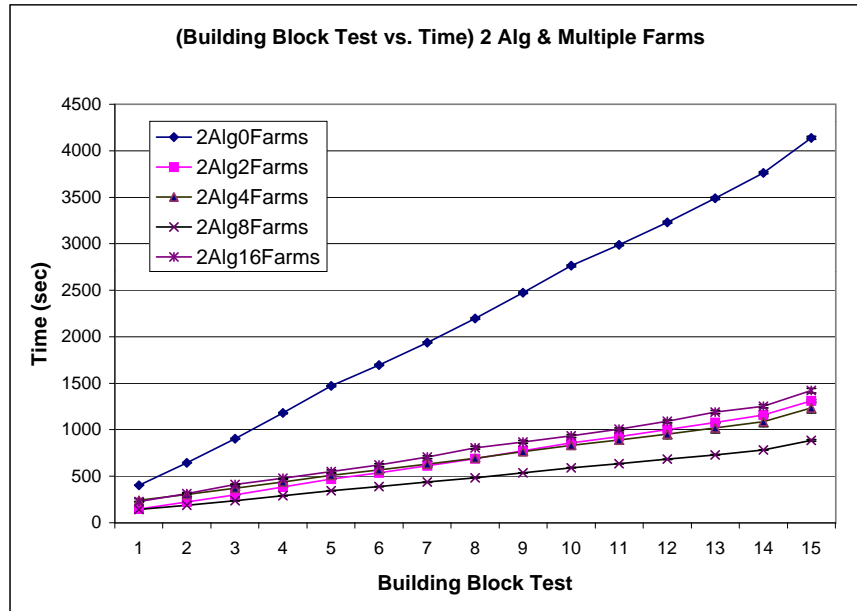


Figure 6.8 Time vs. BB Test for 2 Algorithm Nodes. Validation of fitness values before and after using the Farming model can be found in Appendix M

To validate the results found from the first test, the number of nodes in increased beyond the number tested in Figure 6.7. Figure 6.8 presents the results for two Algorithm nodes tested with a number of different farming nodes. In this testing, the overall population size across all of the Algorithm nodes is equivalent to the population size used in the test of a single Algorithm node. One can see here some of the same conclusions previously drawn: as the BB size is increased, the execution time is also increased. Additionally a significant improvement is noted in modifying the number of farming processors per algorithm node from 0 to 2. A minor improvement is seen in going from 1 to 2 and 2 to 4 farming processors but going from 4 to 8 processors is detrimental. In increasing the number of farming processors to 8 per Algorithm node, results in each farming node being under utilized and hence achieving a worse speedup than other configurations. The serial time was 4140 seconds and the parallel time with 4 compute nodes per farm took 887 seconds yielding a speedup of 4.7. These results illustrate the usefulness of the farming processors for the protein POLY.

6.3 Building Block Size analysis

The building block (BB) analysis is performed in an attempt to identify the building block sizes that result in finding better solutions for POLY. A BB is a partial string representing bits from one, some, or all of the dihedral angles that each chromosome represents [75]. The BBs are not restricted to be contiguous bits from the chromosomes but instead can be non-contiguous bits from the chromosome. Therefore if one purely looks at just one BB it may represent a whole dihedral angle or just various bits of multiple angles.

This analysis covers a variety of BB sizes and compares the results to determine which size produces the best statistical results. One expects a BB size of 35 bits to yield the best due to the alpha helix [12] structure of POLY. Alpha helix proteins are

known *a priori* to have 3.5 residues per turn [8]. The BB ranges chosen for testing included: 16-18, 18-20, 20-22, . . . , and 38-40.

The results of the BB size experiment are presented in Figure 6.9. BB sizes of 30-32 yielded the best results for POLY. Although, this BB size is specific for POLY, it should apply to other proteins having an alpha helix structure. Additionally, BB size 30-32 yielded the best overall fitness value found during all of the BB testing of -140 kcal, which is in the neighborhood of the accepted CHARMM fitness for this protein.

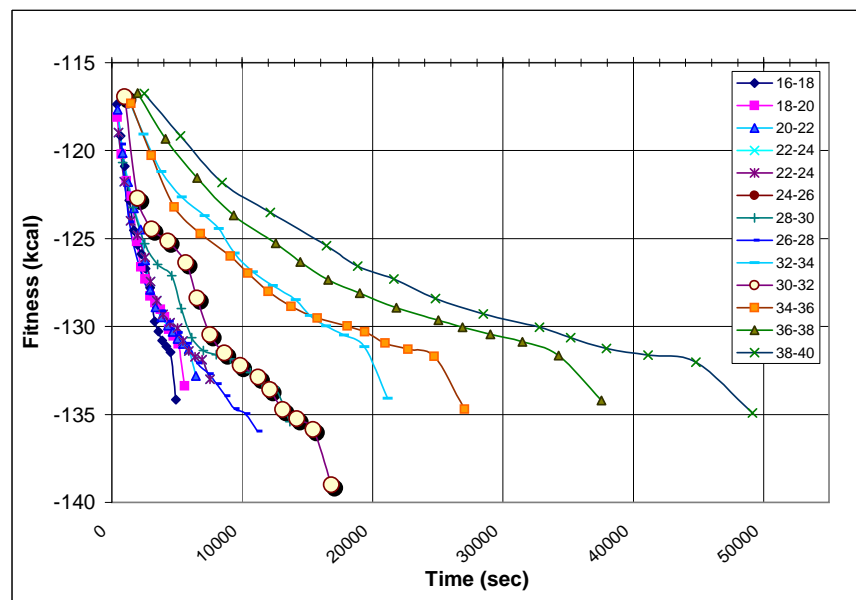


Figure 6.9 Time vs. Building Block Test plot of building block sizes and associated best fitness found from each experiment. Specific computers used in this experiment can be found in Appendix M in Table M.1

Figure 6.9 depicts the average of the results for experiments run 10 times each. Furthermore, it is shown in Section 6.1 that even experiments running closely together still are found to be different using the Kruskal-Wallis and paired observations test; therefore, we can draw from these earlier statistical analogies to infer that the BB size 30-32 is statistically better than the rest. It still should be said that no direct statistical methods are used to validate this conclusion.

6.4 Protein 3D File Generation

The product from experiments run using the PDB file format generation provides the researcher realized conformations of proteins. Figure 6.10 illustrates a good conformation for MET found in during the MOfmGA experiment. The fitness value for MET at this particular conformations is -33 kcal/mol. A visualization of POLY is also shown in this investigation in Figure 6.11. This conformation of POLY was also found during an experiment using the MOfmGA. POLY's fitness is -170 kcal/mol for this conformation.

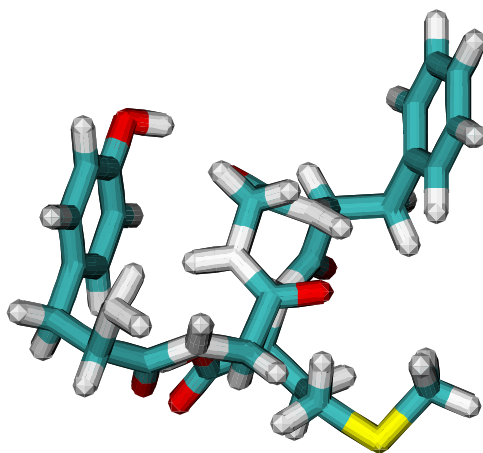


Figure 6.10 Conformation from a PDB file [17]. This protein had a fitness value of -33 kcal/mol.

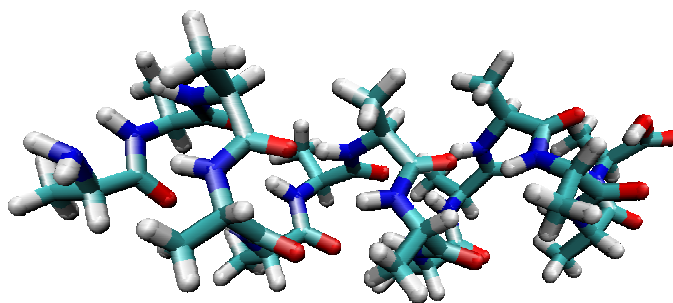


Figure 6.11 Conformation from a PDB file [17]. This protein had a fitness value of -169 kcal/mol.

Included is a graphical representation of the accepted conformation of *Polyalanine*₁₆ (Illustrated in Figure 6.12 – a close relative of *Polyalanine*₁₄). Both of these are

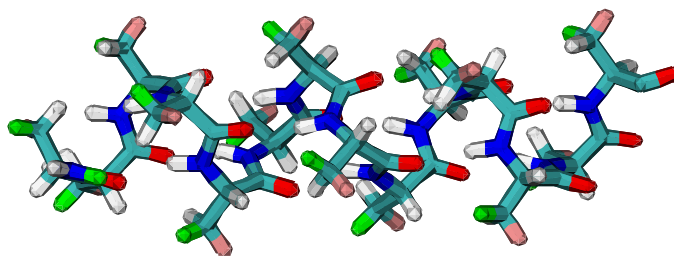


Figure 6.12 Conformation from a PDB file representing the accepted conformation for *Polyalanine*₁₆. Notice that the conformation is nearly the same found in our MOfmGA search (represented in Figure 6.11).

known to form Alpha-helix structures. Notice that the conformations of these two structures are very similar. This visual test can also validate our findings in this manner.

6.5 Multiobjective Experiment

The multiobjective experiment is our feature experiment in this investigation. As discussed in Section 4.2.5 the fitness function is decomposed into two meaningful subsets: Physics (Objective 1) and Chemistry (Objective 2). The Physics subset represents the non-bonded energy functions of the CHARMM fitness model. This objective targets the coulomb interactions and steric anatomy of the protein are kept correct. Additionally, it can be understood that objective 1 focuses on keeping good topology of the protein. Whereas, the Chemistry subset represents the bonded energy functions of the CHARMM fitness model due to the *fixed* model characteristics. This objective helps in keep the Classical description of Chemistry for a protein correct. Moreover, dihedral angle, bond lengths, and bond angle energies are optimized with this objective. For a complete breakdown of the objective functions into energy functions see Section 4.2.5.

The previous competitive template approaches are combined with the MO experiment for it is customary to build on good implementations of algorithms. This reflects the extension of the algorithm during this investigation. The MOfmGA is

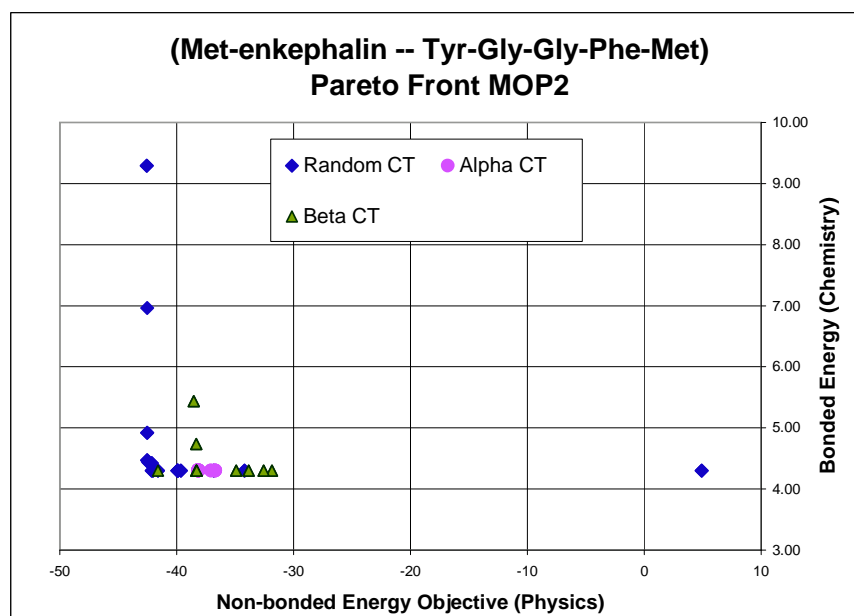


Figure 6.13 [Met]-enkephlan Pareto Front.

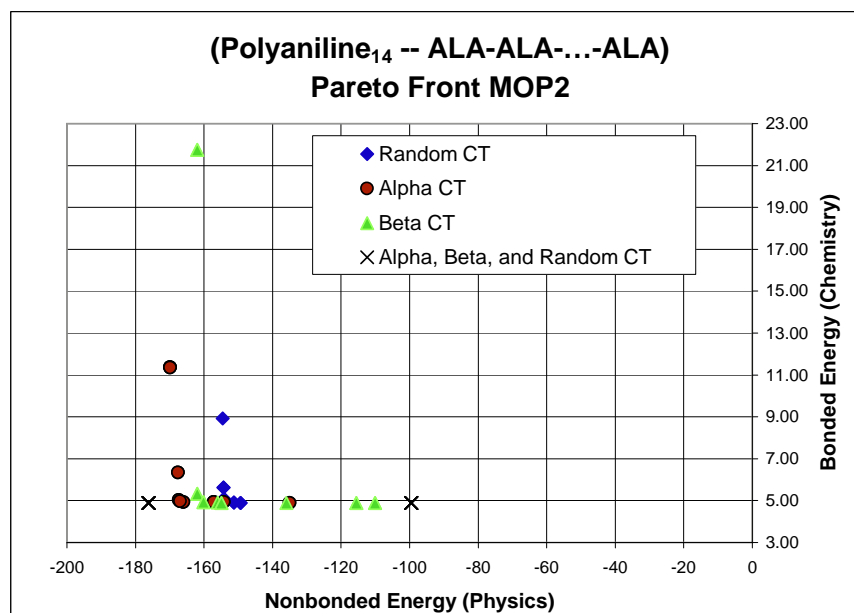


Figure 6.14 *Polyalanine*₁₄ Pareto Front

executed 10 times for each of the experiments in order to provide statistical results. All of the results presented are averaged over 10 runs and the Pareto Front plots are the combined results over the ten runs. Over every run, the following MOfmGA

parameters were kept constant; cut probability = 0.02, splice probability = 1.00, primordial generations = 200, juxtapositional generations = 100, total generations = 300. An input schedule was used to specify sizes of the building blocks the algorithm uses and during which generations BBF occurs. Tests were conducted using both MET, with 240 bit length strings and BB sizes 6-10, and POLY, with 560 bit length strings and BB sizes 16-20.

Figure 6.13 presents the Pareto Front found from the MET testing. In this figure the Random competitive template obtained the best distribution of points along the front, all of the points are Pareto Front members if combined with the other methods, as well as the largest cardinality out of the three competitive template methods tested. This is expected as MET does not contain a secondary structure and hence neither the Alpha nor Beta methods provide better results than the random generation of the competitive template.

Figure 6.14 presents the Pareto Front found from the POLY testing. In this figure the Alpha competitive template method performed the best in terms of the overall distribution of points along the front as well as the cardinality of the Pareto Front set. This is expected since POLY has a Alpha-helix structure; thus, the Alpha competitive template should provide the best results. However, it is found that the multiple competitive template method, which has an Alpha-helix template, performed the best (See Table 6.2).

The CT testing produced “good” results and results that are anticipated considering the structure of the proteins analyzed. The multiobjective implementation of the fmGA compares favorably to the original fmGA results regarding minimum energies. Since the MOfmGA implementation involves decomposing the summation of terms used in the original fmGA, one can sum up the two fitnesses and obtain what the single objective value would be and then make a limited comparison to the original fmGA results. Table 6.2 presents the results of the best found fitness for each of the proteins from the original fmGA testing and the MOfmGA testing. For MET

the MOfmGA finds the best overall fitness value when compared with the original fmGA. In the POLY analysis, the MOfmGA compares favorably to the original.

Table 6.2 Best Fitness Found

	Alpha	Beta	Random	A,R&B
Met	-31.716	-33.191	-34.114	-31.834
MO Met	-33.857	-37.287	-38.047	N/A
Poly	-163.393	-157.203	-159.105	-171.760
MO Poly	-162.246	-156.624	-149.052	-171.314

By Table 6.2 results, it might seem that the MOfmGA did not perform as well as the single objective for the POLY; however, the results obtained by the MOfmGA are more meaningful than that of the single objective. Produced along one side of the Pareto Front is the spread of a topology search and structural chemistry kept along the other. Thus, one knows that the chemistry is more flexible, we might loosen these constraints and decide that the weighted proportion of this objective should be less than the first objective. Furthermore, selection of the correct conformation may be influenced by the decision making making intelligent choices between these two objectives. This met Objective 7 of goals outlined in Section 1.2

6.6 Ramachandran Experiment

The Ramachandran experiment is conducted to take advantage of problem domain information in restricting the search space (not size) for the algorithm. Search space constraining is normally advantageous to a search algorithm; however, this implementation provides better resolution for the feasible area of solutions instead of confining the searchable area. In other words, the search space for all three variables is exactly the same (See Section 4.2.6). In the preliminary results the MOfmGA was executed three times for each of the methods to provide statistical results. All results presented here are averaged over three runs and the plots represent an average of these three runs. The following MOfmGA parameters are kept constant; cut probability = 0.02, splice probability = 1.00, primordial generations = 200, juxtapo-

sitional generations = 200, total generations = 400. An input schedule was used to specify sizes of the building blocks the algorithm uses and during which generations BBF occurs. Tests were conducted using only POLY, with 560 bit length strings and BB sizes 20-24. Furthermore, the n_a variable (population sizing variable is defined in Equation 3.8) is set at 100 and, for efficiency of getting results, only a single objective and a single randomly generated competitive template is employed.

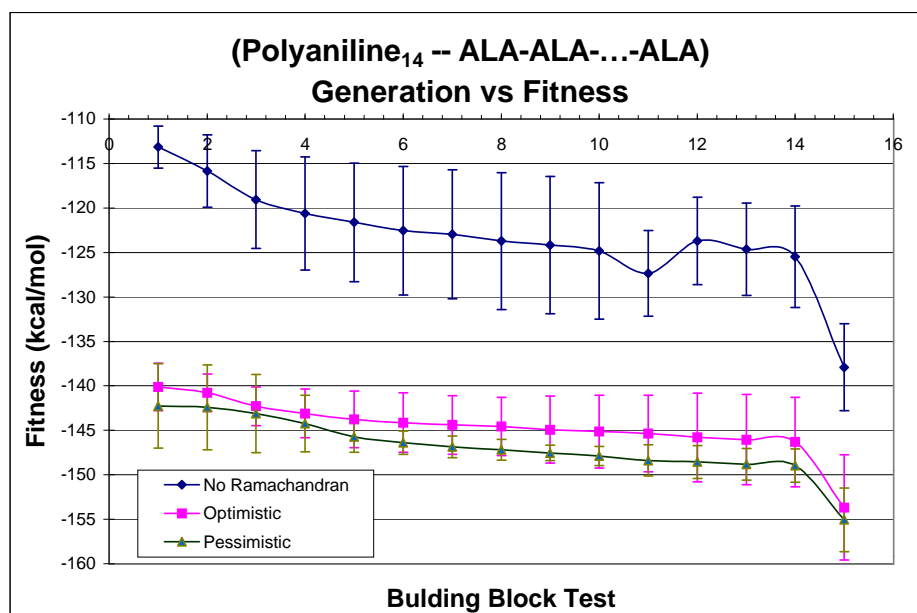


Figure 6.15 Building Block Test vs. Fitness plot of results for an experiment using no, pessimistic and optimistic Ramachandran plots on the protein POLY. See Appendix C for the restrictions applied to the landscape for each different method.

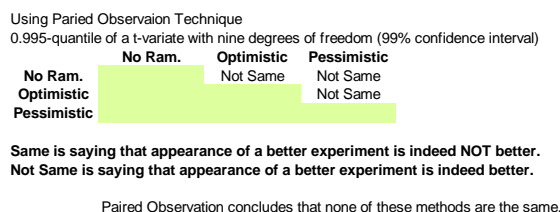


Figure 6.16 Summary of results after conducting the paired observation difference test found in [47] on the experiment using no, pessimistic and optimistic Ramachandran plots on the protein POLY.

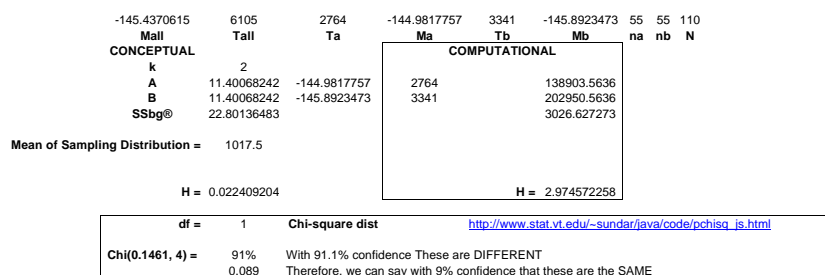


Figure 6.17 Summary of results after conducting the Kruskal-Wallis test on the results from the pessimistic and optimistic implementation of the Ramachandran Plots. Note: the test concludes 91% confidence that these are different; furthermore, the pessimistic constraints more effective.

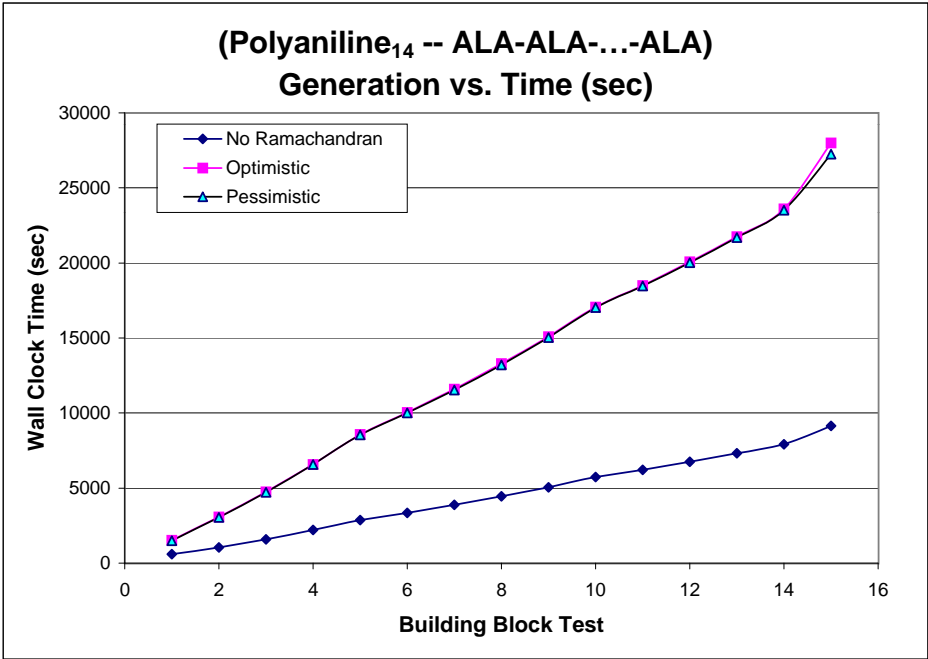


Figure 6.18 Building Block Test vs. Time to Complete plot of results for an experiment using no, pesimistic and optimistic Ramachandran plots on the protein POLY.

	Dihedral(°)	Dihedral (RAD)	Cartesian Coordinate ($\text{\AA} = 10^{-10}m$)
All atoms	1177.8	20.56	102.0
Backbone atoms	153.0	2.67	184.0

Table 6.3 RMS difference calculations for POLY.

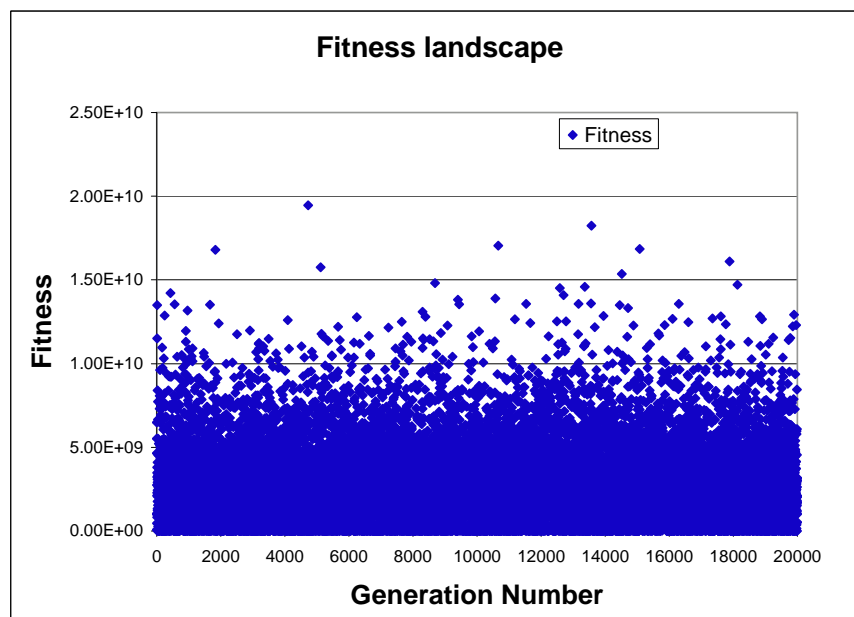


Figure 6.19 Generation of a Randomly created structure vs. Fitness

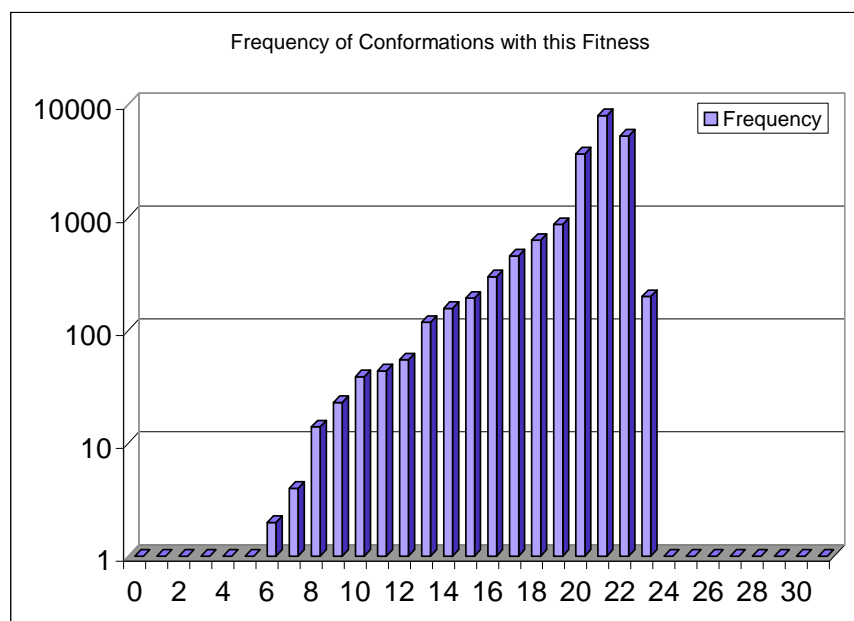


Figure 6.20 $\ln(\text{Fitness})$ vs. Frequency Found (Randomly generated conformations)

6.6.0.1 Effectiveness. Figure 6.15 illustrates the results of the Ramachandran experiment. It is clear from the graph that both the Optimistic and Pes-

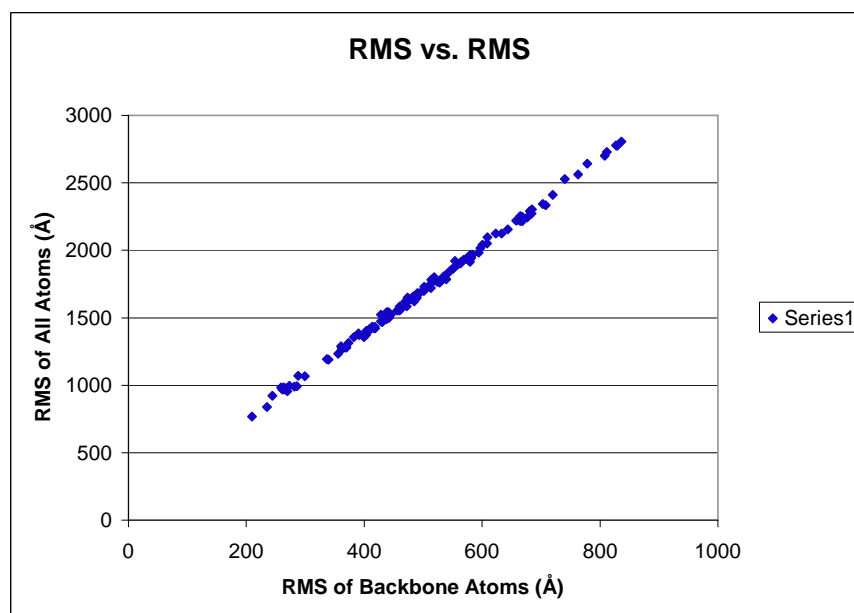


Figure 6.21 RMS Cartesian Coordinate difference for all atoms vs. RMS Cartesian Coordinate difference for only backbone atoms. This is for validations of the RMS calculation and to ensure that we have a linearity between these two differences. Both RMS differences are measured in Angstroms (Å). Where $1 \text{ Å} = 10^{-10}$

Atoms	Dihedral(o)	Dihedral(RAD)	Deerman [20]	Grid Monte Carlo(o)
All	737.9	12.9	17.124 (unk units)	n/a
Backbone	480.7	8.4	n/a	37.4
Chi	411.3	7.2	n/a	11.3

Table 6.4 RMS difference calculations for MET.

simistic Ramachandran constraints achieve better results than the none Ramachandran implementation. Statistically, the paired observation test in Figure 6.16 and the Kruskal-Wallis test in Figure 6.17 both confirm that each of these methods are different than one another; however, the student t-test has the Optimistic and Pessimistic implementations as being the same. Just the same, the Kruskal-Wallis test is our standard test to use on data about which we know nothing of the distribution. Furthermore, the pessimistic Ramachandran values are more effective than both the optimistic and non-use of the Ramachandran plots.

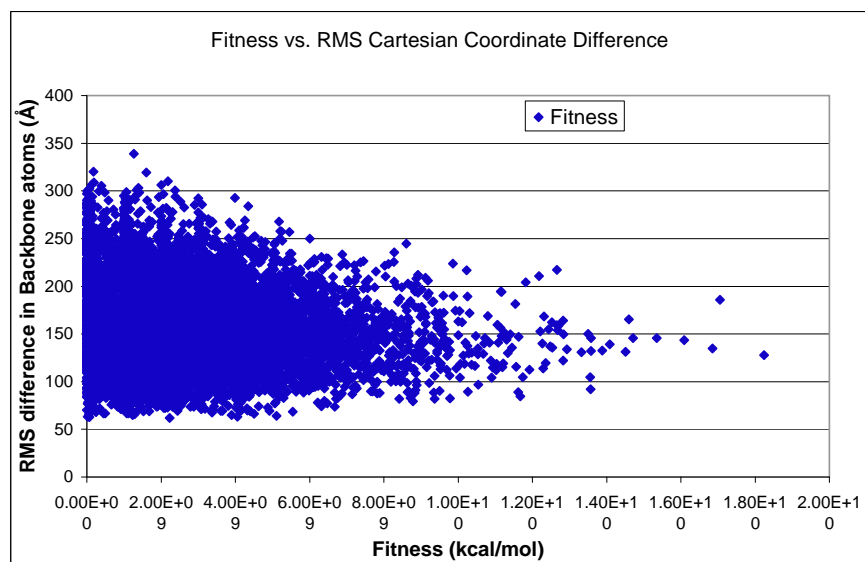


Figure 6.22 Fitness vs. RMS Cartesian Coordinate difference (kcal/mol vs. Angstroms (\AA)). (Fitness evaluation on randomly generated conformations)

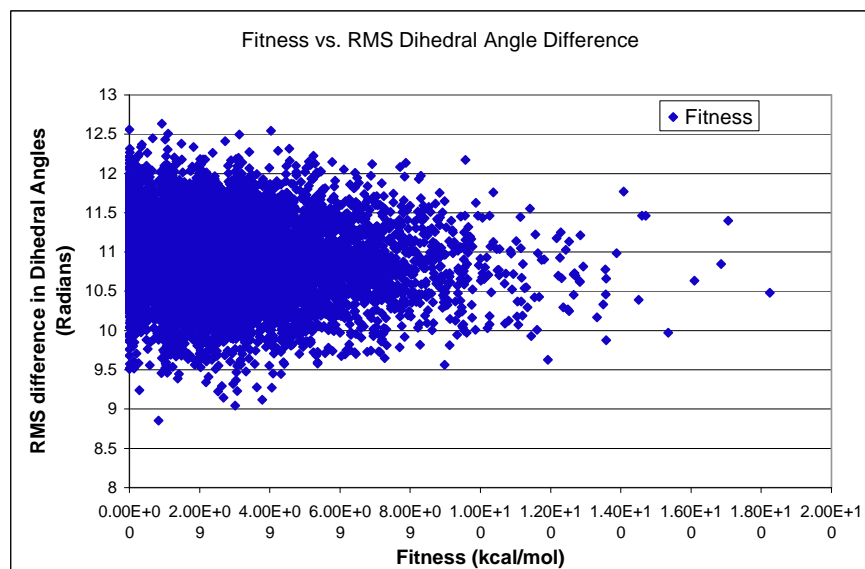


Figure 6.23 Fitness vs. RMS Dihedral Angle difference. (kcal/mol vs. Radians) (Fitness evaluation on randomly generated conformations)

6.6.0.2 Efficiency. The effectiveness of using the Ramachandran plots do not come at cost in efficiency. The mapping cost three times that of the none Ramachandran implementation. This is presented in Figure 6.18. Notice that

the mapping for both the Optimistic and Pessimistic implementation takes exactly the same cost in time; therefore, if one were to use these constraints, the research must choose the Pessimistic values because it is statistically more effective but costs the same in time.

6.7 RMS difference

Found angle	TRUE	Angle type	Residue
105.5	156	phi	tyr
-169.8	-86	psi	tyr
42.9	180	omega	tyr
-55.5	84	phi	gly1
135	-155	psi	gly1
-38.7	-177	omega	gly1
103	-74	phi	gly2
-109.3	84	psi	gly2
-38	169	omega	gly2
81.6	19	phi	phe
-163.5	-137	psi	phe
50.3	-170	omega	phe
8.8	160	phi	met
-124.8	-173	x1	tyr
-88.6	-101	x2	tyr
134	73.7	x1	phe
-54.9	108.7	x2	phe
80.2	53	x1	met
-29.5	175	x2	met
-126.2	180	x3	met
-16.5	-164	psi	met
9.5	-174	omega	met

Table 6.5 Angles found for MET at ~ -38 kcal/mol.

The RMS difference calculation defined in Section 4.2.7 is a measure to see how close a found conformation is to an accepted true conformation. All such calculations are accomplished on the POLY protein because it is known that MET folds in to many different acceptable conformations. Because it is thought that the landscape should correspond to RMS differences a Fitness versus RMS plot is generated. The

Found	TRUE	Angle type	Residue	Found	TRUE	Angle type	Residue
116.4	0	phi	Ala1	313.9	319.9	psi	Ala10
313.3	180	psi	Ala1	186.3	180	omega	Ala10
175.7	180	omega	Ala1	286.2	295.1	phi	Ala11
284.8	295	phi	Ala2	321	319.9	psi	Ala11
322.8	320	psi	Ala2	183.2	180	omega	Ala11
173.3	180	omega	Ala2	294.2	295	phi	Ala12
294.2	295	phi	Ala3	322	319.9	psi	Ala12
317.1	320	psi	Ala3	193.3	180	omega	Ala12
195.1	180	omega	Ala3	279.5	295	phi	Ala13
295.3	295	phi	Ala4	336.5	320	psi	Ala13
309.8	320	psi	Ala4	181.4	180	omega	Ala13
189.4	180	omega	Ala4	278.8	295	phi	Ala14
281.3	295	phi	Ala5	56.9	300.6	chi	Ala1
309.7	320	psi	Ala5	55.6	300.6	chi	Ala2
200.4	180	omega	Ala5	174.7	300.6	chi	Ala3
282	295	phi	Ala6	61.5	300.6	chi	Ala4
319.2	320	psi	Ala6	170.1	300.6	chi	Ala5
186.7	180	omega	Ala6	317.5	300.6	chi	Ala6
289	295	phi	Ala7	81.1	300.6	chi	Ala7
320	319.9	psi	Ala7	302.7	300.6	chi	Ala8
186.7	180	omega	Ala7	170.5	300.6	chi	Ala9
290.1	295.1	phi	Ala8	60.2	300.7	chi	Ala10
322.3	319.9	psi	Ala8	186.7	300.6	chi	Ala11
183.9	180	omega	Ala8	314.3	300.6	chi	Ala12
293.6	295	phi	Ala9	295.3	300.6	chi	Ala13
309.7	320	psi	Ala9	296	300.6	chi	Ala14
209.2	180	omega	Ala9	175.9	181.1	psi	Ala14
251.4	295	phi	Ala10	314	140.6	omega	Ala14

Table 6.6 Angles found for POLY at ~ -170 kcal/mol.

Cartesian Coordinate difference is illustrated in Figure 6.22. This plot was expected to yield a linear association between RMS difference and fitness values calculated for particular conformations. Furthermore, it was to be used to validate our fitness model. Unfortunately, this is not the case. There is no linear correlation between calculated fitness values and Cartesian Coordinate RMS difference. The reason for this is due to the dihedral angle mapping back to the Cartesian coordinate system. There may be good conformations (low fitness values) found with bad Cartesian

Coordinate RMS differences. This is because the first dihedral angle might be off target, lending all following atoms attached to that dihedral angle to also be off target. Therefore, it is not true that the Cartesian Coordinate RMS Difference can weight the merit of a protein structure. That said, there might be inference to identify that the Dihedral RMS Difference is correlated with Fitness values found for particular conformation. However, this too turned out to be a fallacy. Figure 6.23 illustrates again that the fitness is not linearly associated to the RMS Dihedral angle difference. The reason for this result is for the roughness of the landscape of the PSP problem. This is illustrated in Figure 6.19. Additionally, previous researchers found this same phenomenon [74][20]. However, to quantify the distribution another plot (Figure 6.20) is provided to see the frequency of the natural log of the fitness for POLY. This met Objective 5 of goals outlined in Section 1.2

The RMS difference of the best found POLY structure from the accepted true structure of POLY can be found in Table 6.3. The RMS differences for MET as well as RMS difference results from previous research can be found in Table 6.3. Table 6.5 illustrates the angles of the best MET structure found in this investigation.

6.8 Comparing to other Research

This investigation compares favorably with previous research. Overall best results POLY -171.76 kcal/mol is the best found since Kaiser using the Regal software when an overall best of -351.76 kcal/mol was found [50]. The most recent study by Michaud [74] found values of -152.56 kcal/mol – using the same type of GA. Moreover, the MO approach found the best ever results with MET – finding a structure at -38.047 kcal/mol. This beats the previously found best result of -36.362 by Gates [32] whom also used the same type of GA. The multiobjective approach has shown that it can yield more meaningful results and always acquires the same if not better overall results than previous methods.

6.9 *Summary*

This Chapter discussed the results of experiments investigated in this Thesis effort. Statistical analysis has been completed to validate that these are in fact better results than previously obtained with the same algorithm. The Multiple Competitive template, Farming Model, Building Block Size, Ramachandran, Protein 3D File Generation and Mutliobjective experiments have all show favorable solutions. However, the RMS vs. Fitness plots did not validate our fitness model as we expected. Further studies of both are suggested. The next Chapter discusses the conclusions of this investigation and recommendations for future research.

VII. *Conclusions and Recommendations*

This research investigation has been a balance of study in three fields: biochemistry, Evolutionary Algorithms, and High Performance Computing. The computer being the medium from which the tool, our MOfmGA, is applied to the biochemistry problem. It is necessary to understand facets of each field in order to make positive contributions to this *Grand Challenge Problem*.

This thesis investigation draws from each of the above fields in showing progression of the new innovative algorithm, *MOfmGA*, to solve the PSP problem. The development of the MOfmGA began with the implementation of a multiple and pan-metic competitive template mechanism to make it more effective [18] (See Sections 4.2.1 and 6.1). The algorithm was modified to scale its efficiency to 4.7 times a serial run time (See Sections 4.2.2 and 6.2). Algorithm development required a major re-write to prepare for the implementation of the multiobjective approach. Software Engineering practices were followed to ease future modifications, like Ramachandran Plot constraints and RMS integration (See Sections 4.2.6, 6.6, 4.2.7, and 6.7). The new algorithm has the capabilities to be single and multiple objective (See Sections 4.2.5 and 6.5) and run with single and multiple competitive templates all configurable. Also the algorithm now provides for optimistic and pessimistic Ramachandran (per residue) constraints and calculation of RMS dihedral and Cartesian coordinate differences from accepted true PDB and Z matrix files of the selected protein.

Computational results support our hypothesis that the MO version of the fmGA produces better results than the previous. This is presented in Section 6.5. Also the Random competitive template scheme performs well in cases where the protein does not contain a secondary structure and a competitive template method that includes the structure of the protein tested performs the best.

7.1 Recommendations

Future work with the MO part of the algorithm should look at other protein structures such as those presented in Table 5.6. Also addressed should be the incorporation of a sharing mechanism to provide a better distribution of points along the Pareto Front [75]. Furthermore, incorporation of a computational steering device should be considered as well as the complete analysis of current fitness function – its limitations and possible expansions or replacement with newer versions of the CHARMM energy model. A re-evaluation of the parallel nature of the MOfmGA and how it can be implemented with MPI2 (threads) needs to be accomplished to keep up with future network and computer advances in architecture and technology [23].

Future research also needs to take a step forward to compete in the Critical Assessment of techniques for protein Structure Prediction (CASP) experiments. In being able to compete at the Protein Structure Prediction Centers, the MOfmGA needs to be modified to easily import new proteins and seamlessly execute a search for the conformation. This modification requires a front-end interface to accomplish file conversion and configuration file building before algorithm execution.

7.1.1 Summary. The main goal of this research has been reached. The birth of a new multiobjective algorithm having extensive capabilities and flexibility to solving the PSP problem is found. Additionally, all secondary goals of this investigation are also achieved. New innovative mechanisms are added to both the fmGA and MOfmGA; and it is, statistically, shown to be better than previous research. New Ramachandran, per-residue, constraints were integrated into the algorithm having also the expected positive impact. A multiple competitive template model is implemented and found to be advantageous. A study of efficiency is validated as to increased algorithm efficiency of the pfmGA and a building block size analysis favored finding secondary structure is complete. Lastly, a RMS difference

of both dihedral angles and cartesian coordinates is also coded into the MOfmGA to give the researcher feedback on the real time RMS distance values from accepted true conformation of found solutions.

The scheduled approach for this research is complete. An improved understand of the PSP problem using new technology and supporting research is satisfied from background and lateral research in the biochemistry and EA arena. Development of a working knowledge of parallel programming concepts for application to the PSP problem domain and algorithm domain are achieved. This is demonstrated by the implementation of the farming model using group-wise communication between nodes. Extensive studies of EA strategies are covered in this thesis investigation. A demonstration of having the ability to re-write an entire fmGA displays an archived working knowledge of the subject material. Biochemistry understanding is also accomplished through the integration of Ramachandran (per residue) plots and RMS difference calculations. Visualization techniques emphasizing the sparseness of good fitness values is present as well as 3D conformations generation found from semi-optimal solutions. Finally, three different statistical methods were applied to the found data to determine the merit of applied innovative mechanisms.

All this said, the projected goals and objectives were met. A new method for discovering good solutions has been found; however, fine tuning of this algorithm is still required. By no means has the PSP been solved, yet this work represents favorable contribution to research in this field of study.

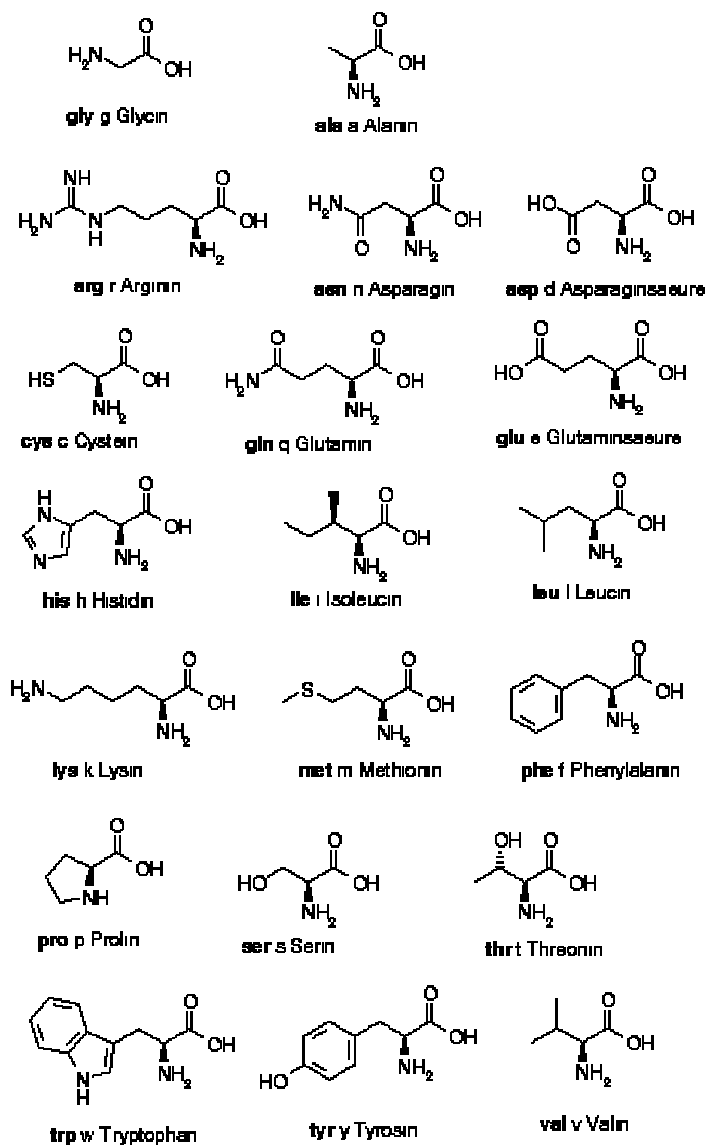
Appendix A. Chemical Formulas for Amino Acids

Amino Acids		
Name	Abbr.	Linear structure formula
=====		
<u>Alanine</u>	ala a	$\text{CH}_3\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Arginine</u>	arg r	$\text{HN}=\text{C}(\text{NH}_2)\text{-NH-}(\text{CH}_2)_3\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Asparagine</u>	asn n	$\text{H}_2\text{N-CO-CH}_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Aspartic acid</u>	asp d	$\text{HOOC-CH}_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Cysteine</u>	cys c	$\text{HS-CH}_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Glutamine</u>	gln q	$\text{H}_2\text{N-CO-}(\text{CH}_2)_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Glutamic acid</u>	glu e	$\text{HOOC-}(\text{CH}_2)_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Glycine</u>	gly g	$\text{NH}_2\text{-CH}_2\text{-COOH}$
<u>Histidine</u>	his h	$\text{NH-CH=N-CH=C-CH}_2\text{-CH}(\text{NH}_2)\text{-COOH}$ _____
<u>Isoleucine</u>	ile i	$\text{CH}_3\text{-CH}_2\text{-CH}(\text{CH}_3)\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Leucine</u>	leu l	$(\text{CH}_3)_2\text{-CH-CH}_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Lysine</u>	lys k	$\text{H}_2\text{N-}(\text{CH}_2)_4\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Methionine</u>	met m	$\text{CH}_3\text{-S-}(\text{CH}_2)_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Phenylalanine</u>	phe f	$\text{Ph-CH}_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Proline</u>	pro p	$\text{NH-}(\text{CH}_2)_3\text{-CH-COOH}$ _____
<u>Serine</u>	ser s	$\text{HO-CH}_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Threonine</u>	thr t	$\text{CH}_3\text{-CH}(\text{OH})\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Tryptophan</u>	trp w	$\text{Ph-NH-CH=C-CH}_2\text{-CH}(\text{NH}_2)\text{-COOH}$ _____
<u>Tyrosine</u>	tyr y	$\text{HO-p-Ph-CH}_2\text{-CH}(\text{NH}_2)\text{-COOH}$
<u>Valine</u>	val v	$(\text{CH}_3)_2\text{-CH-CH}(\text{NH}_2)\text{-COOH}$

Figure A.1 List of linear structure formula for Amino Acids. [30]

Appendix B. 2D Conformation/Chemical Formulation for Amino

Acids



INSTITUTE OF CHEMISTRY
Department of Biology, Chemistry, Pharmacy • FU Berlin

Figure B.1 Conformation/Chemical formulation for Amino Acids.

Appendix C. Ramachandran Worksheets

C.1 Alanine

Alanine, ALA, Ramachandran Worksheet

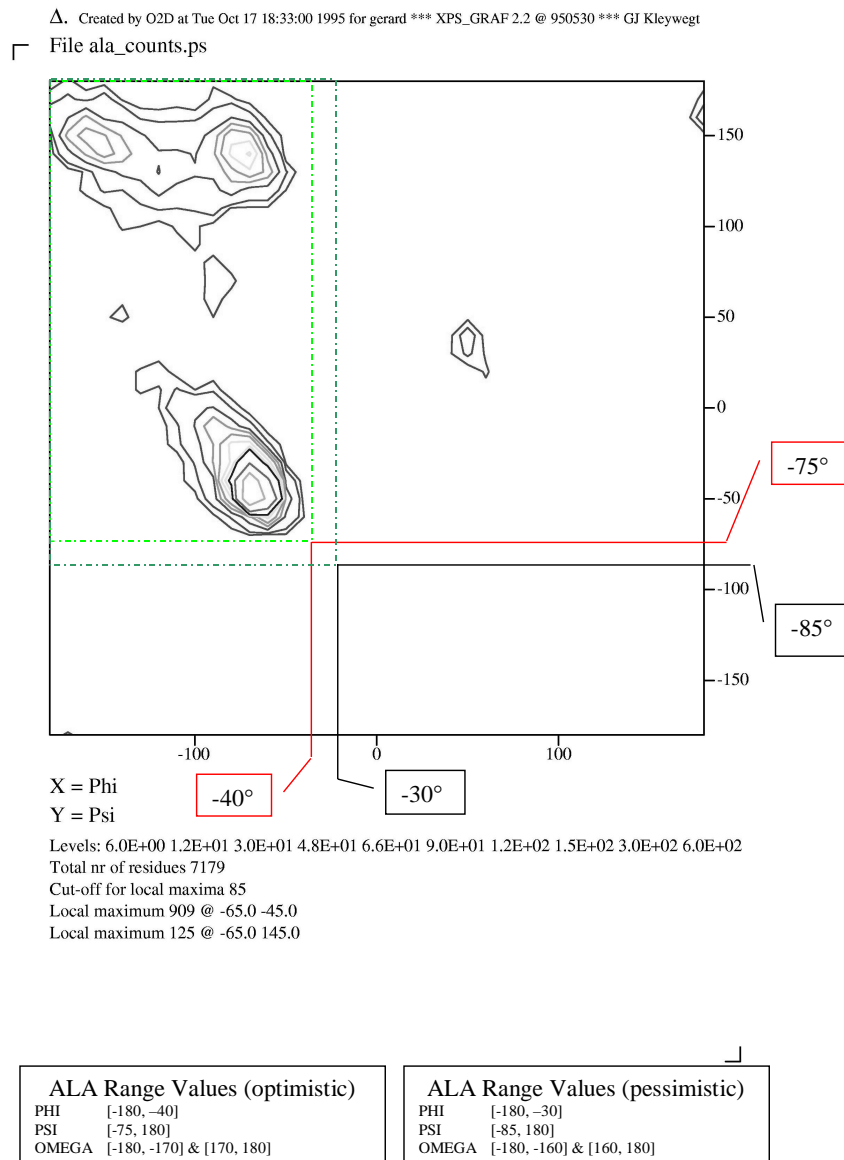


Figure C.1 Alanine Ramachandran Worksheet.

C.2 Arginine

Arginine, ARG, Ramachandran Worksheet

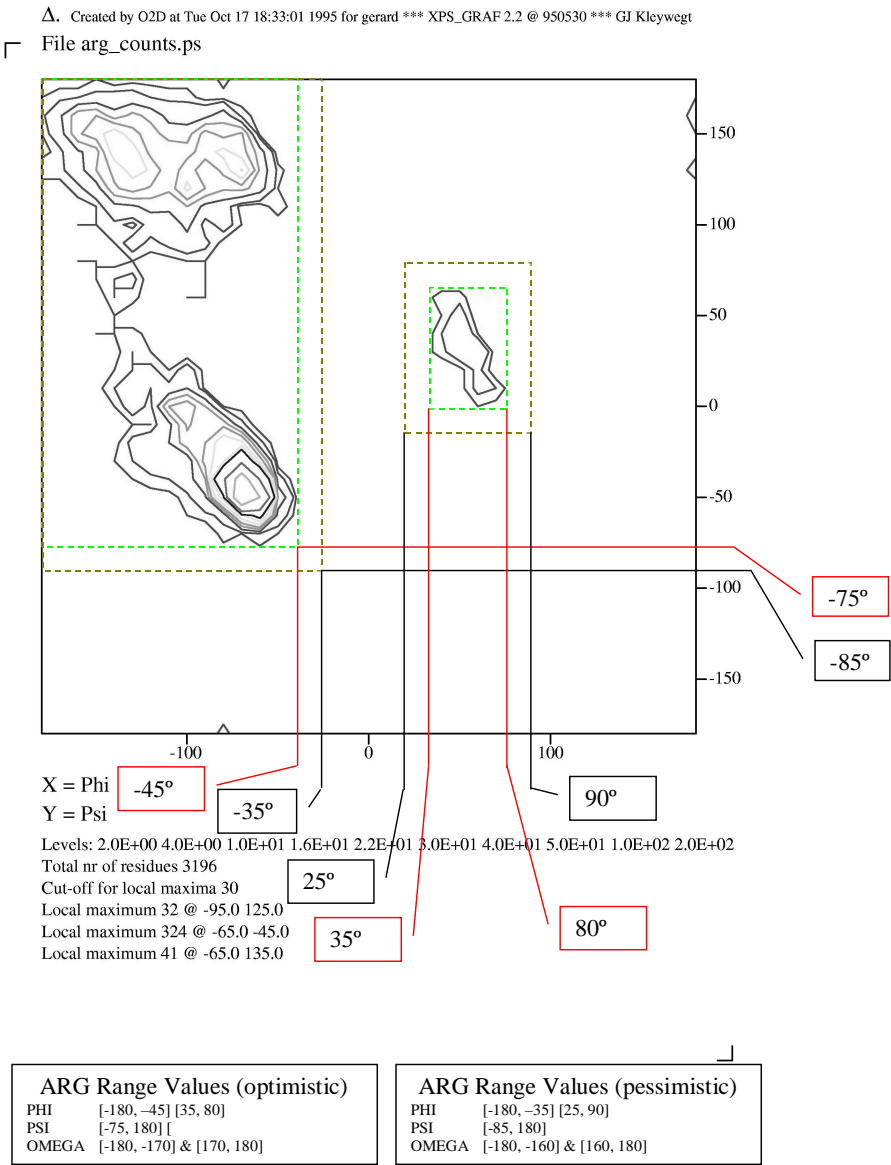


Figure C.2 Arginine Ramachandran Worksheet.

C.3 Asparagine

Asparagine, ASN, Ramachandran Worksheet

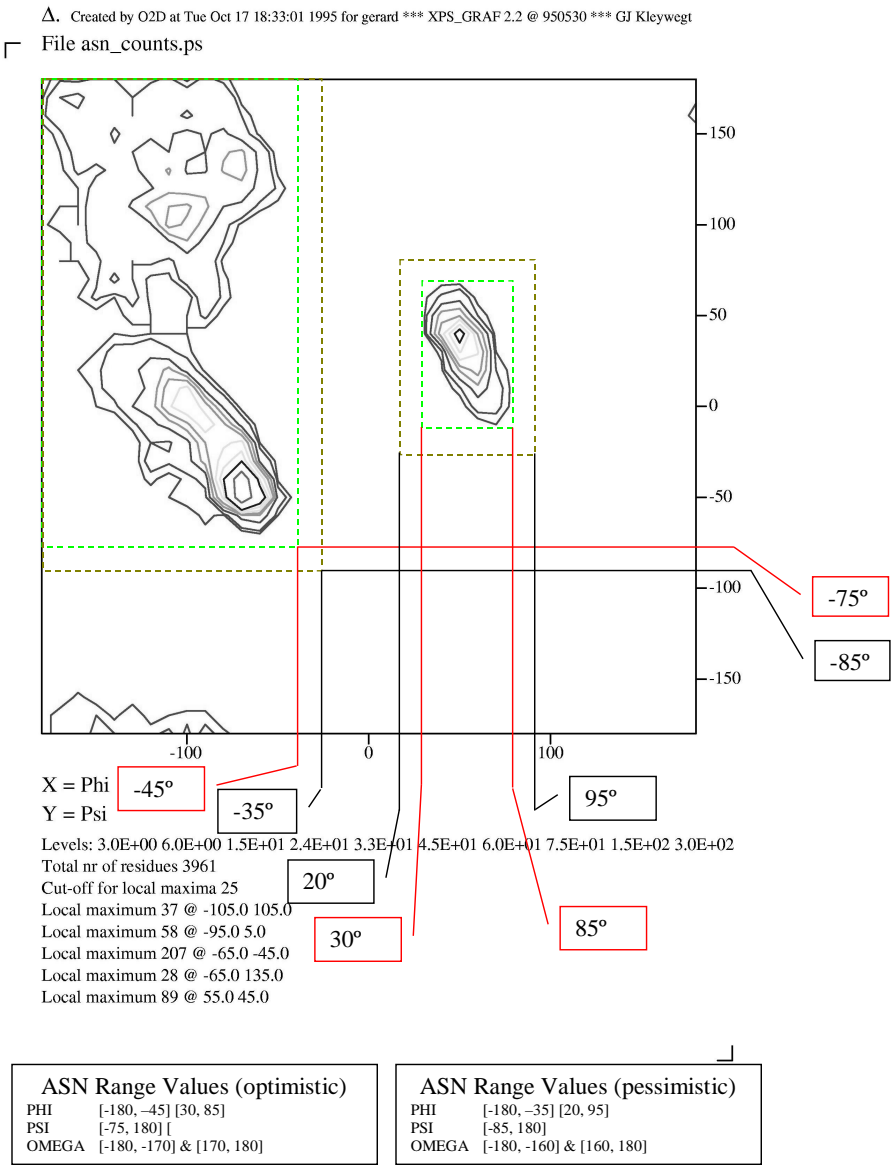


Figure C.3 Asparagine Ramachandran Worksheet.

C.4 Aspartic acid

Aspartic acid, ASP, Ramachandran Worksheet

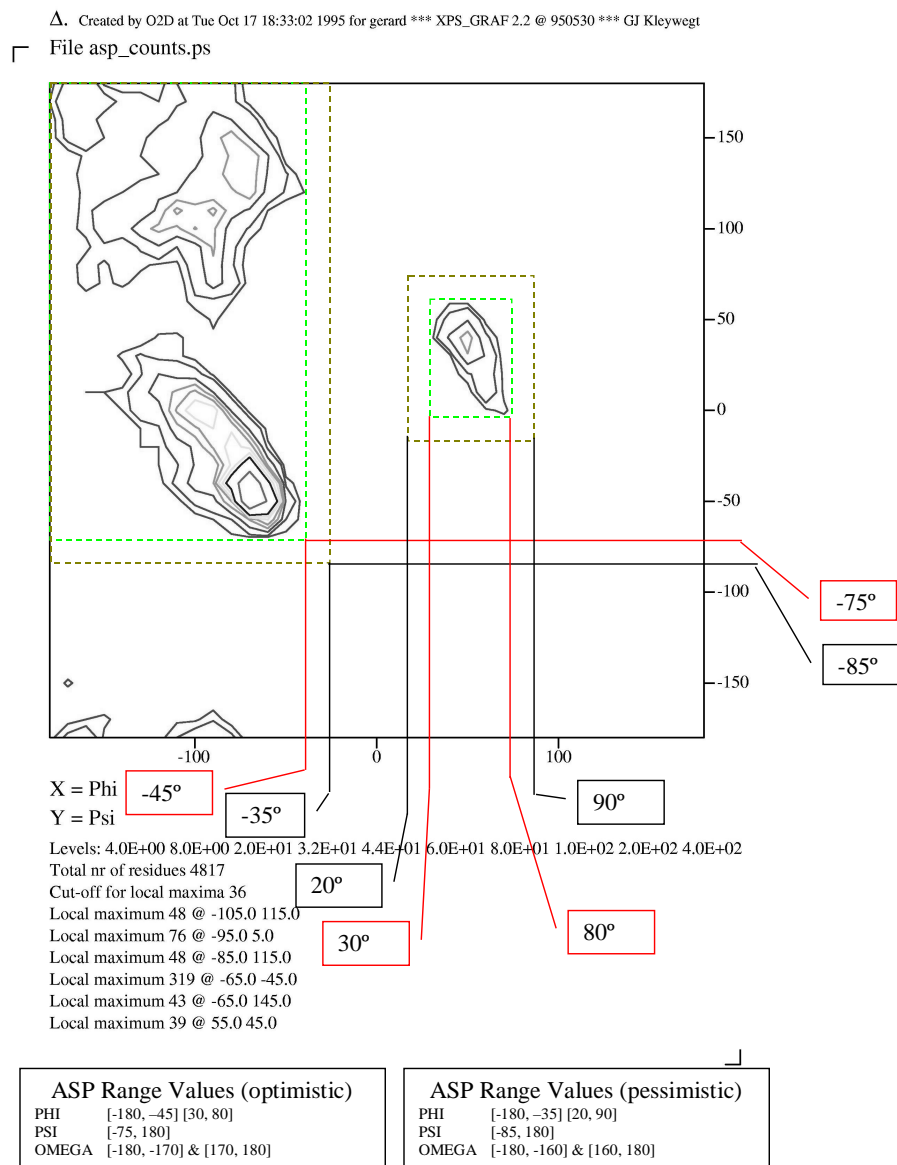


Figure C.4 Aspartic acid Ramachandran Worksheet.

C.5 Cysteine

Cysteine, CYS, Ramachandran Worksheet

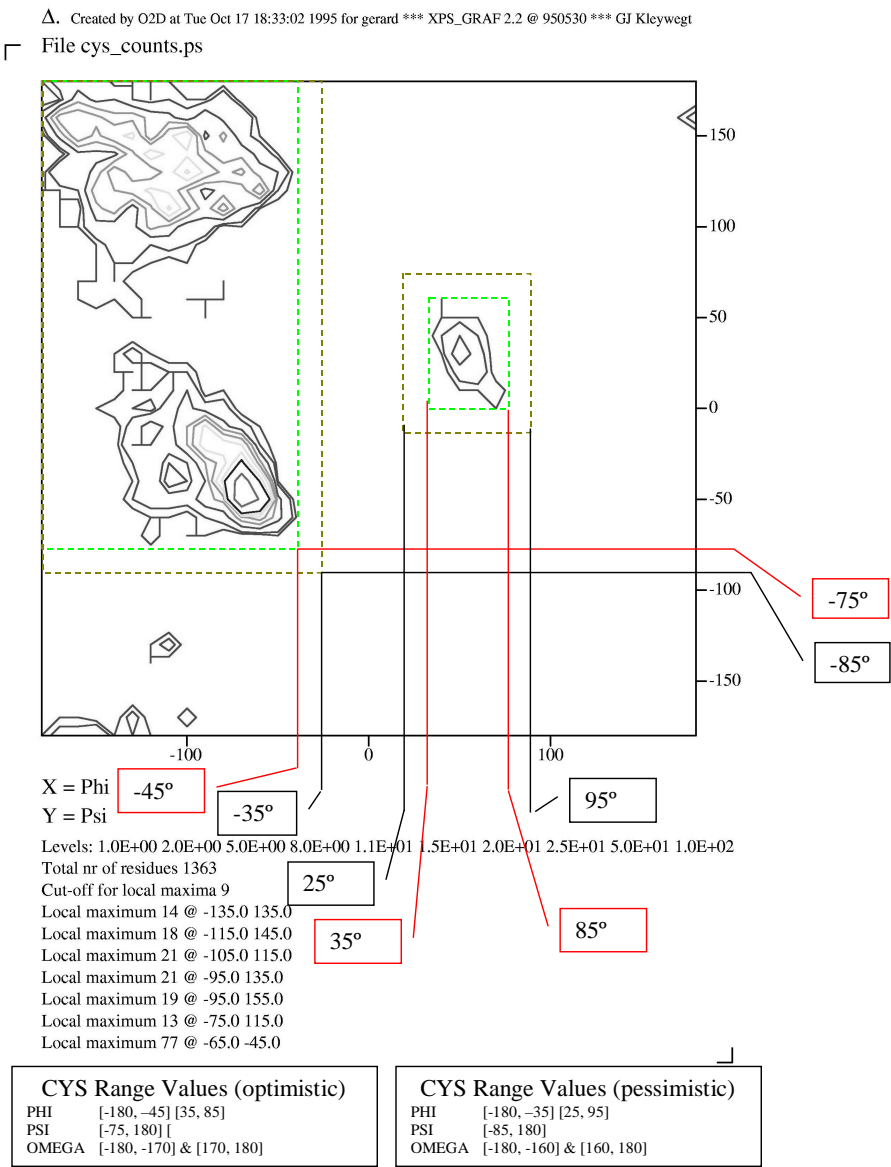


Figure C.5 Cysteine Ramachandran Worksheet.

C.6 Glutamine

Glutamine, GLN, Ramachandran Worksheet

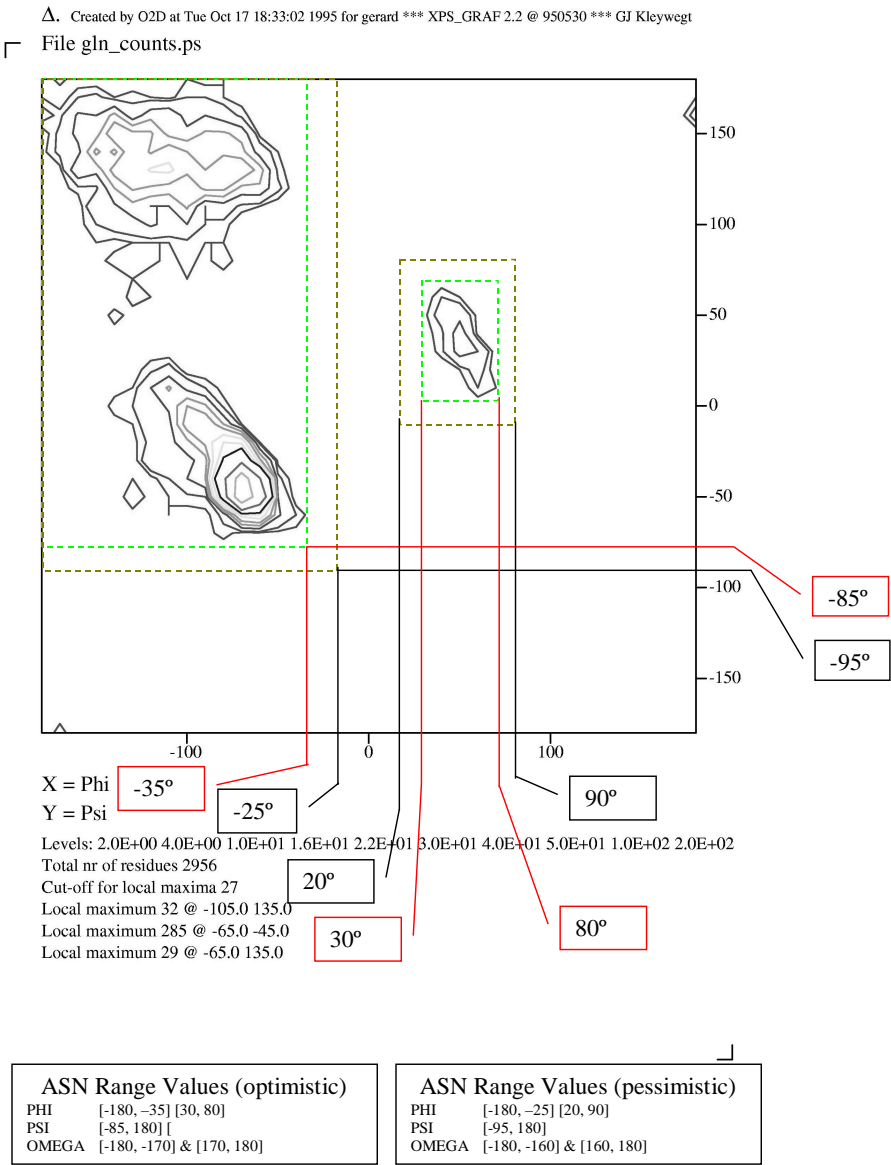


Figure C.6 Glutamine Ramachandran Worksheet.

C.7 Glutamic

Glutamic Acid, GLU, Ramachandran Worksheet

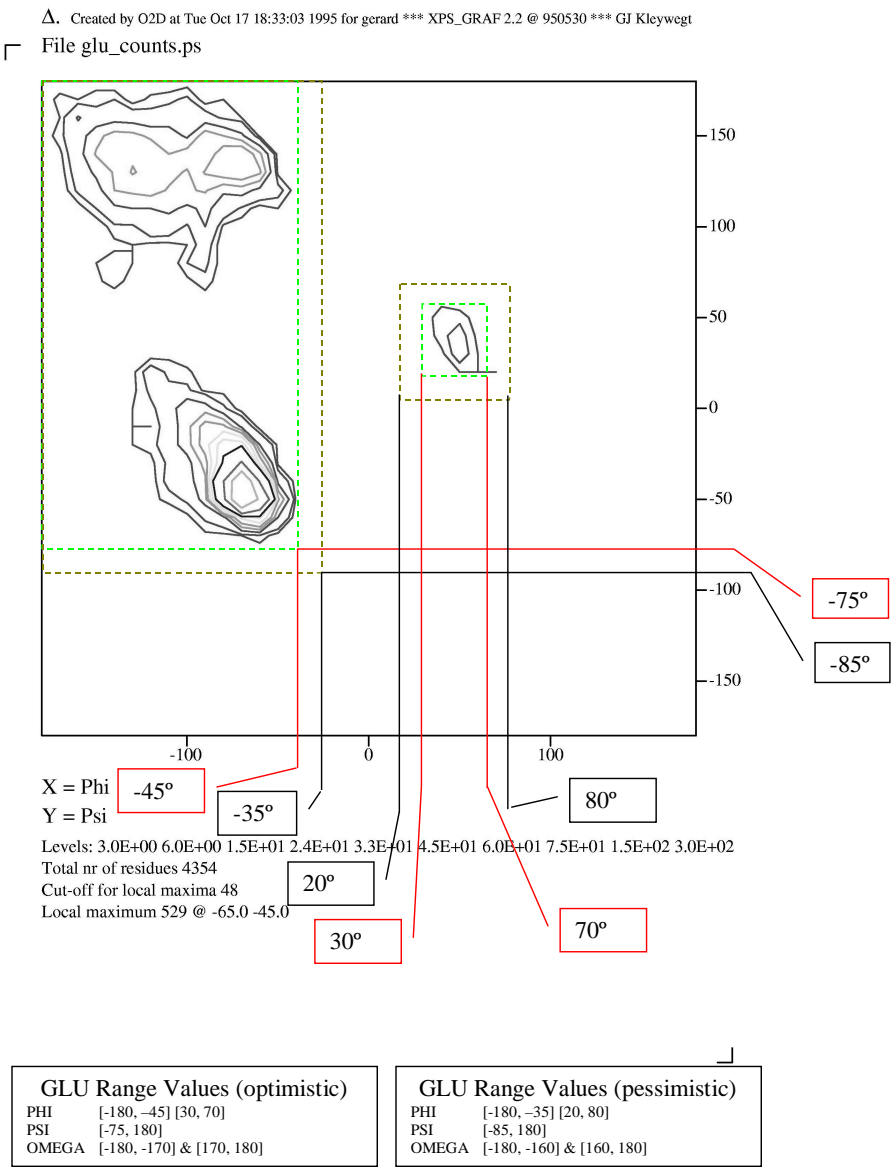


Figure C.7 Glutamic Ramachandran Worksheet.

C.8 Glycine

Glycine, GLY, Ramachandran Worksheet

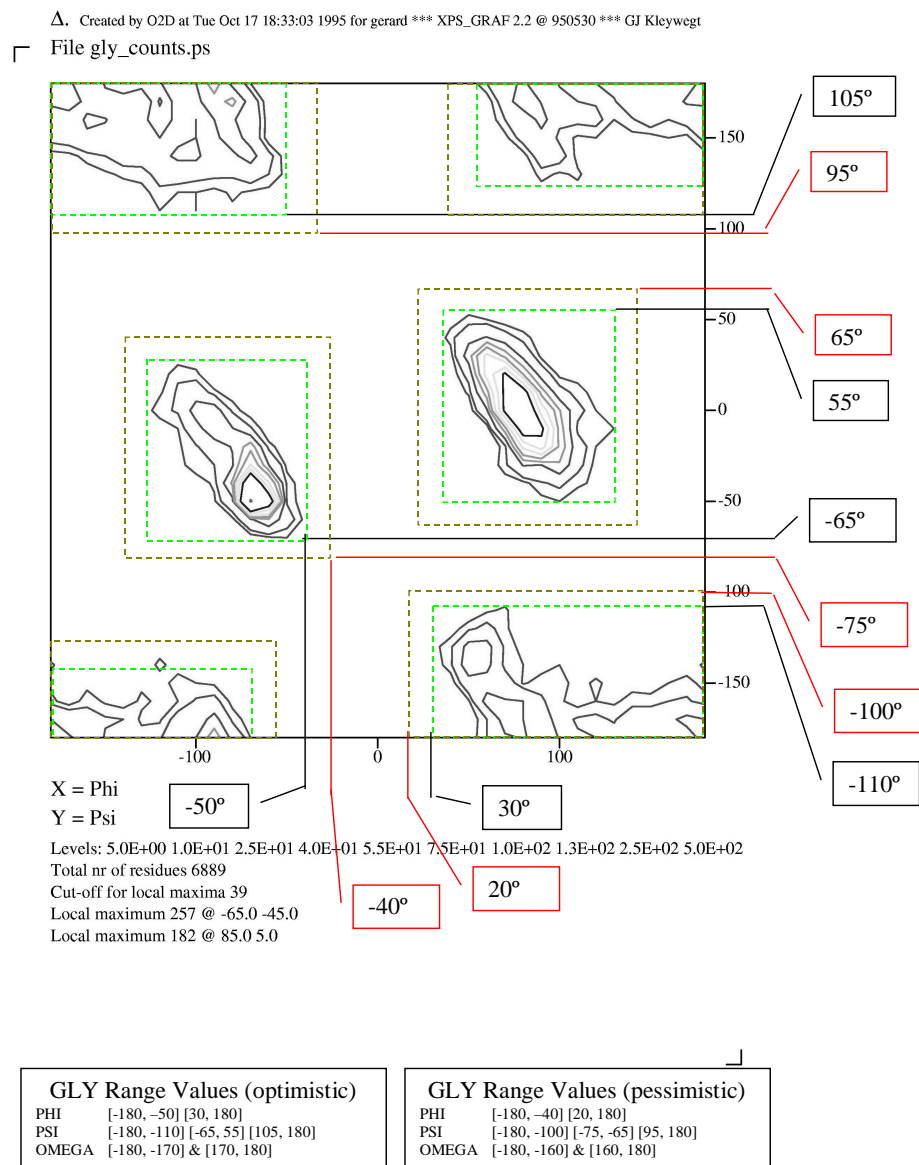


Figure C.8 Glycine Ramachandran Worksheet.

C.9 Histidine

Histidine, HIS, Ramachandran Worksheet

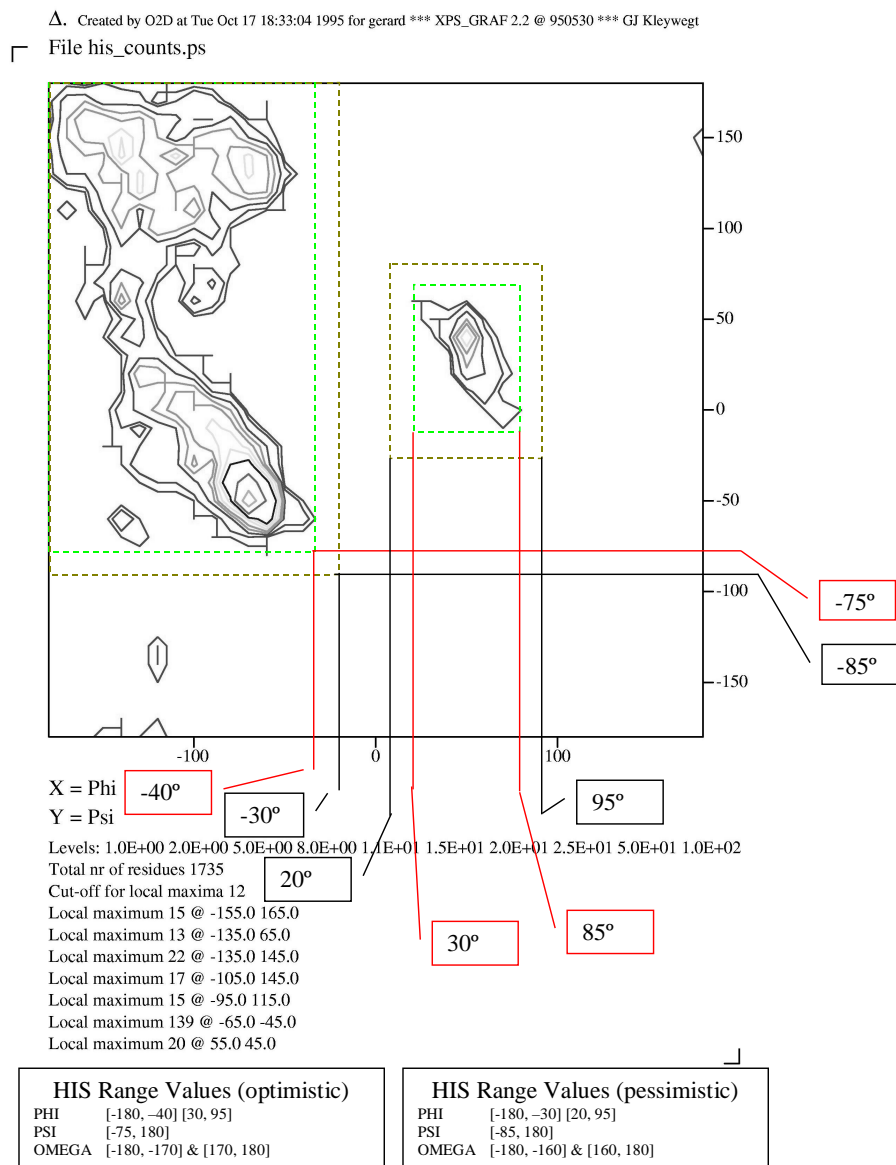


Figure C.9 Histidine Ramachandran Worksheet.

C.10 Isoleucine

Isoleucine, ILE, Ramachandran Worksheet

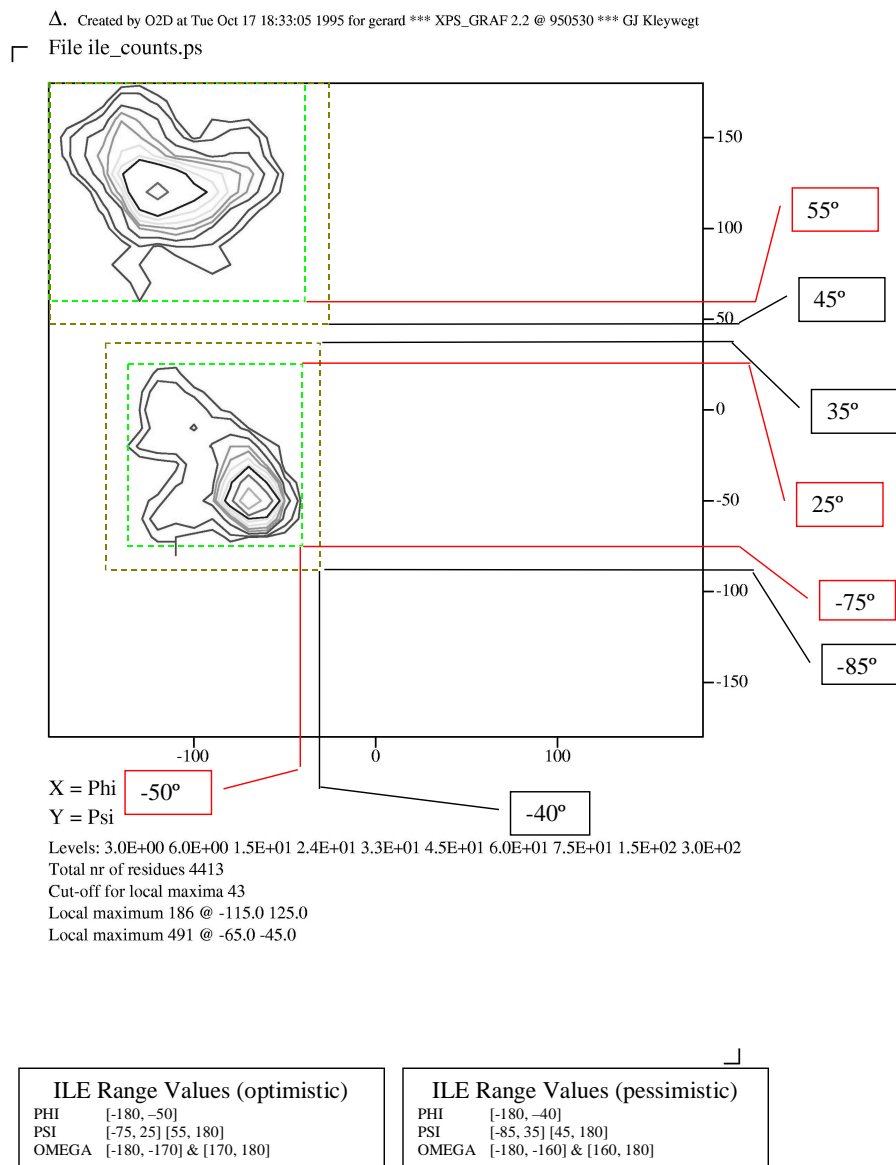


Figure C.10 Isoleucine Ramachandran Worksheet.

C.11 Leucine

Leucine, LEU, Ramachandran Worksheet

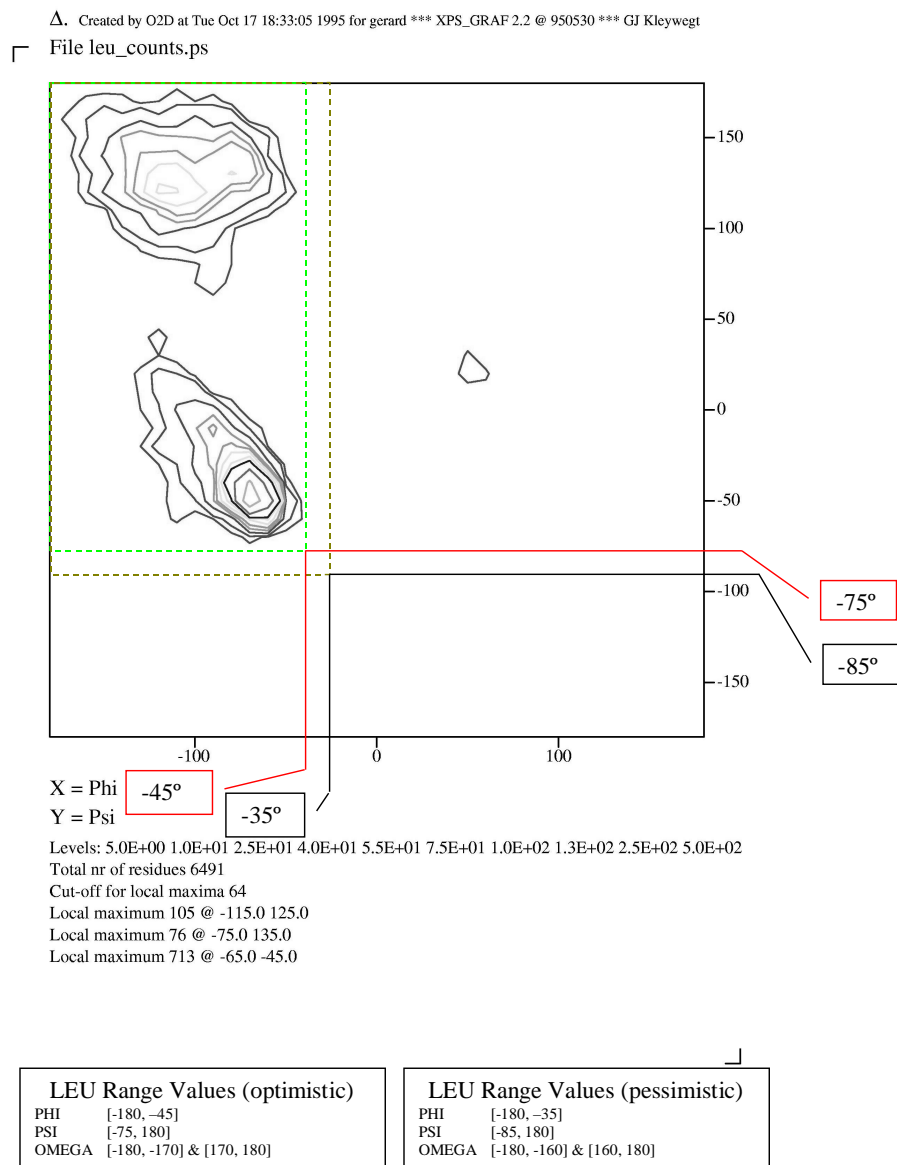


Figure C.11 Leucine Ramachandran Worksheet.

C.12 Lysine

Lysine, LYS, Ramachandran Worksheet

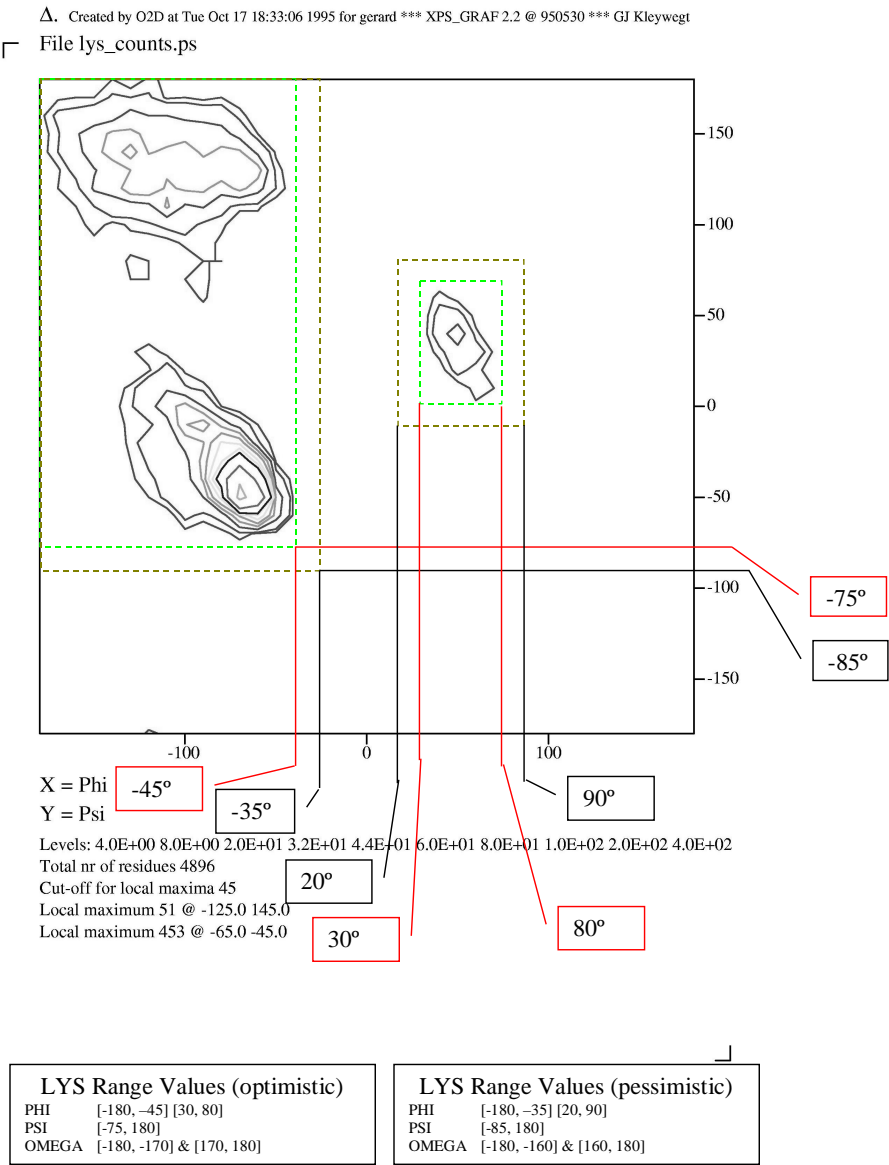


Figure C.12 Lysine Ramachandran Worksheet.

C.13 Methionine

Methionine, MET, Ramachandran Worksheet

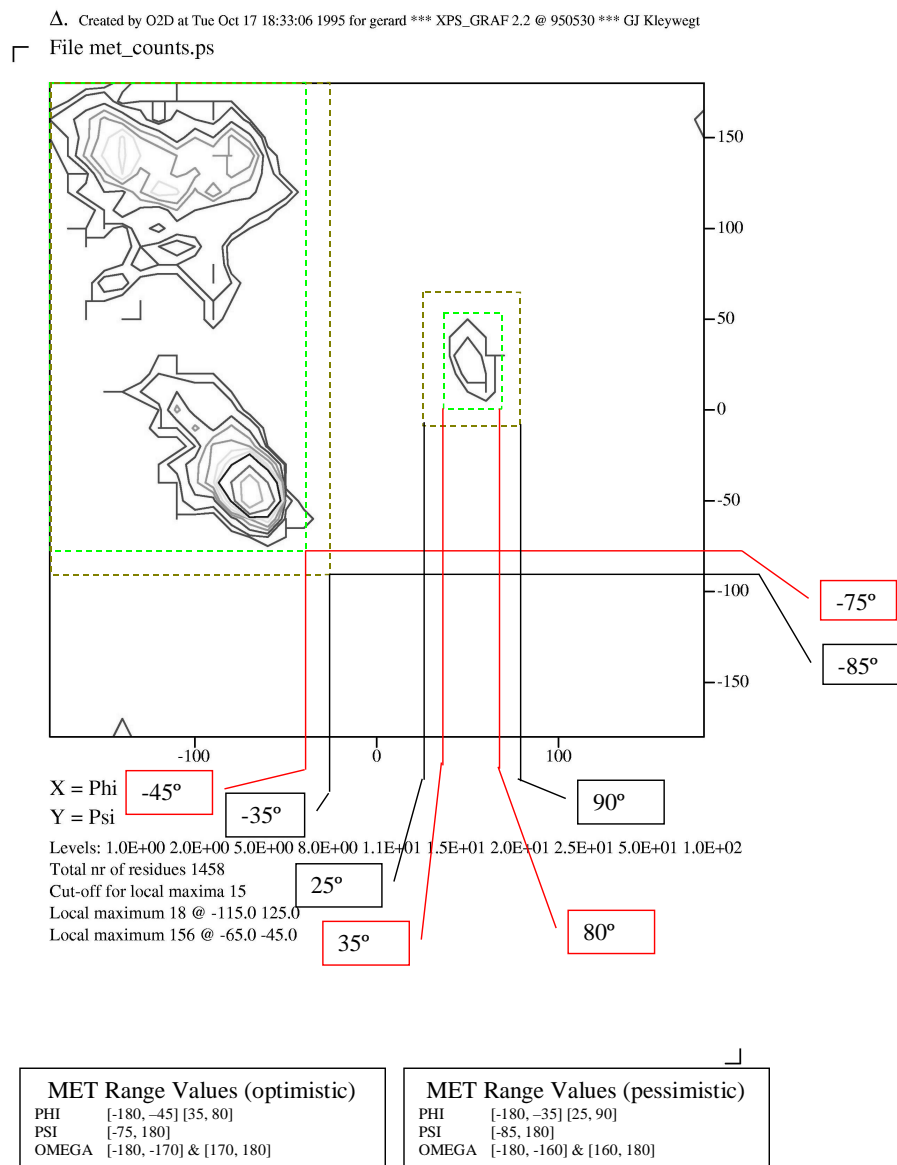


Figure C.13 Methionine Ramachandran Worksheet.

C.14 Phenylalanine

Phenylalanine, PHE, Ramachandran Worksheet

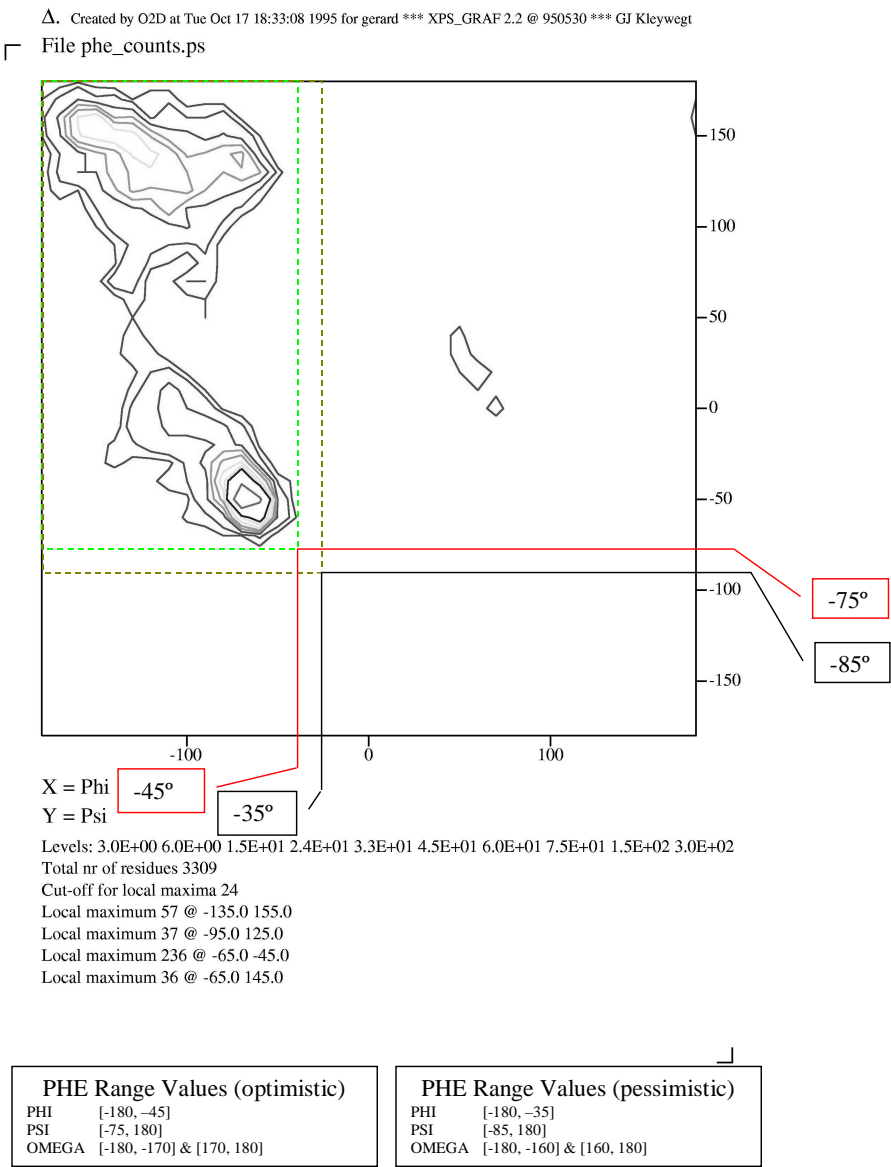


Figure C.14 Phenylalanine Ramachandran Worksheet.

C.15 Proline

Proline, PRO, Ramachandran Worksheet

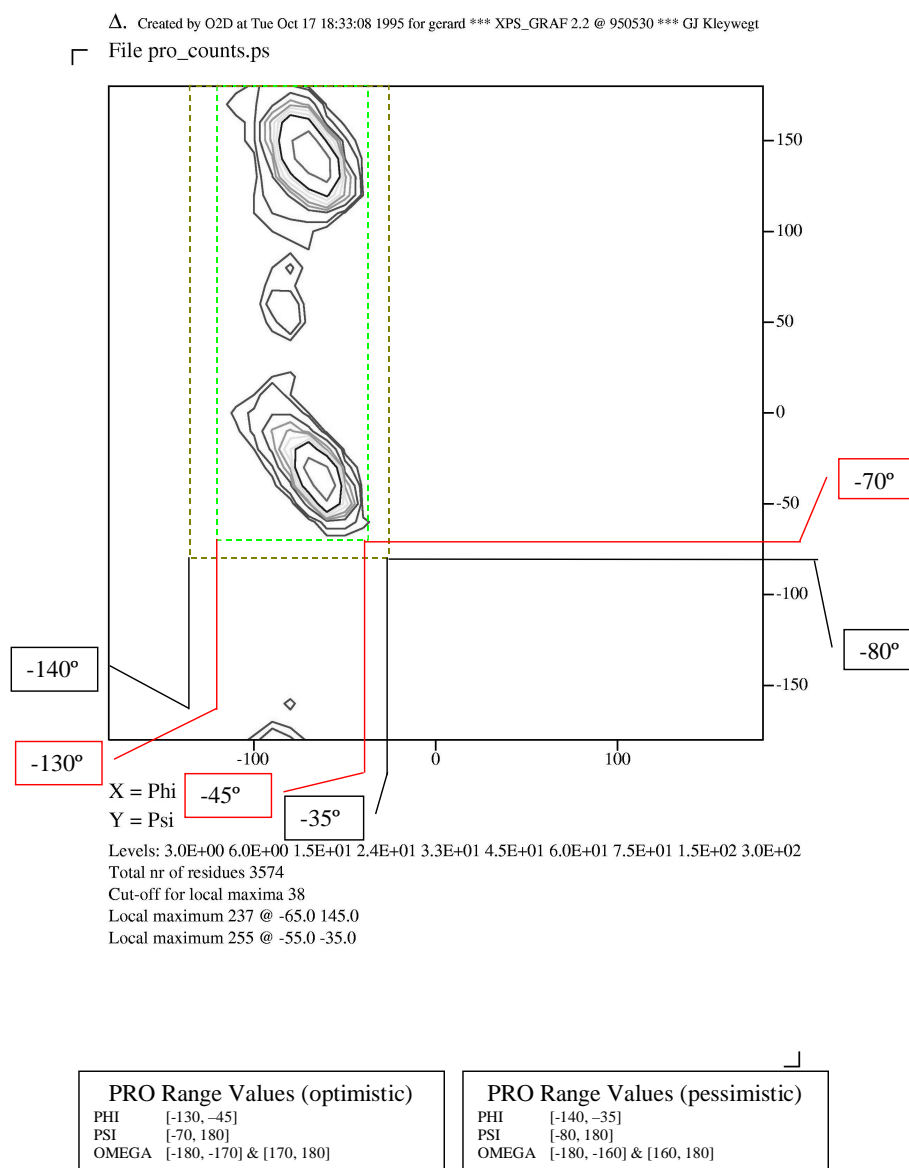


Figure C.15 Proline Ramachandran Worksheet.

C.16 Serine

Serine, SER, Ramachandran Worksheet

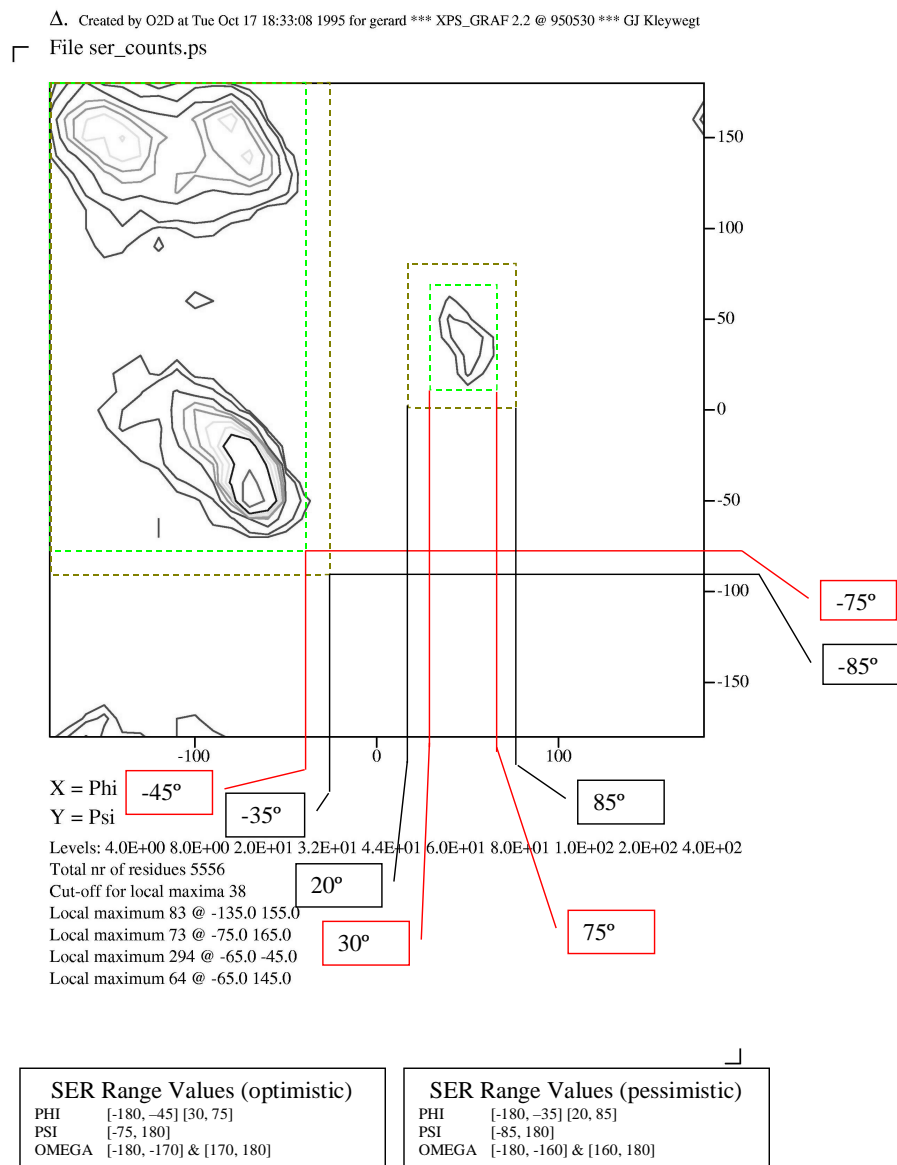


Figure C.16 Serine Ramachandran Worksheet.

C.17 Threonine

Threonine, THR, Ramachandran Worksheet

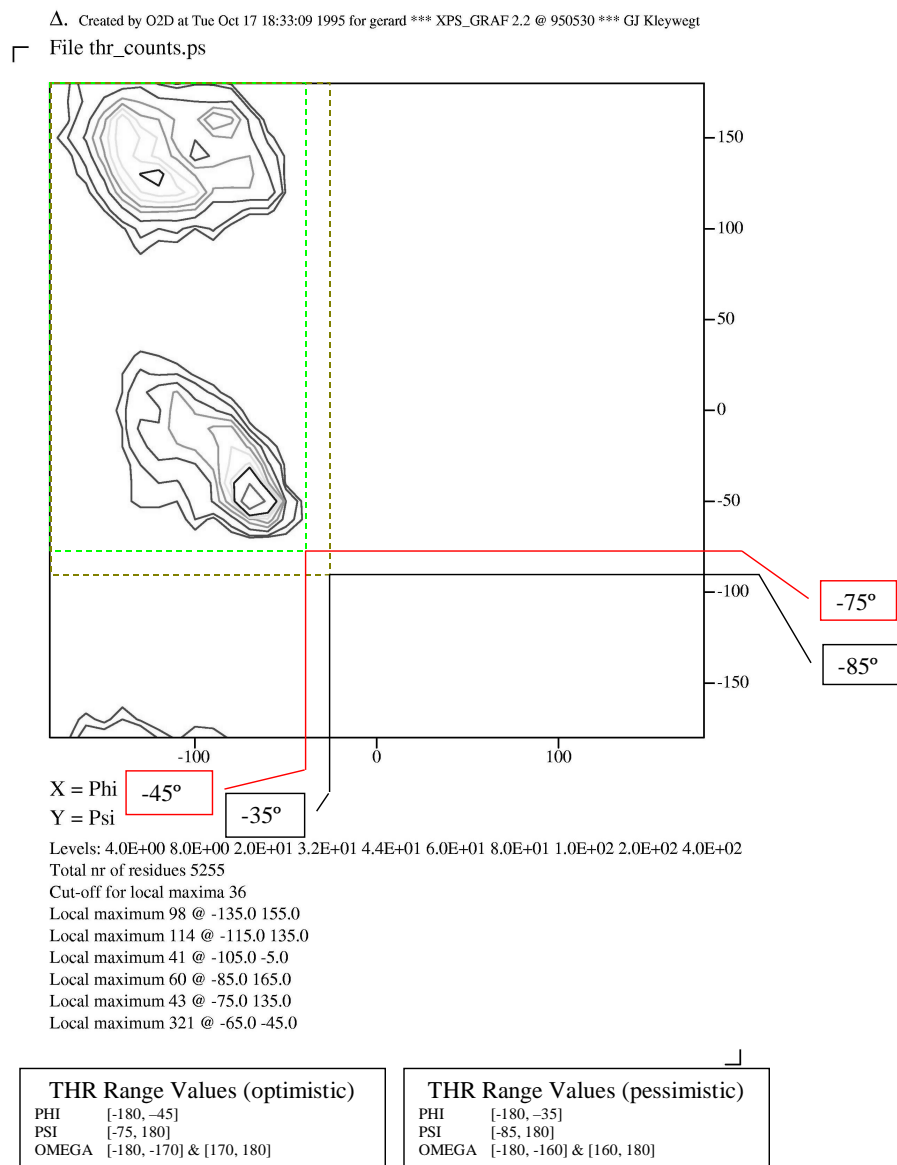


Figure C.17 Threonine Ramachandran Worksheet.

C.18 Tryptophan

Tryptophan, TRP, Ramachandran Worksheet

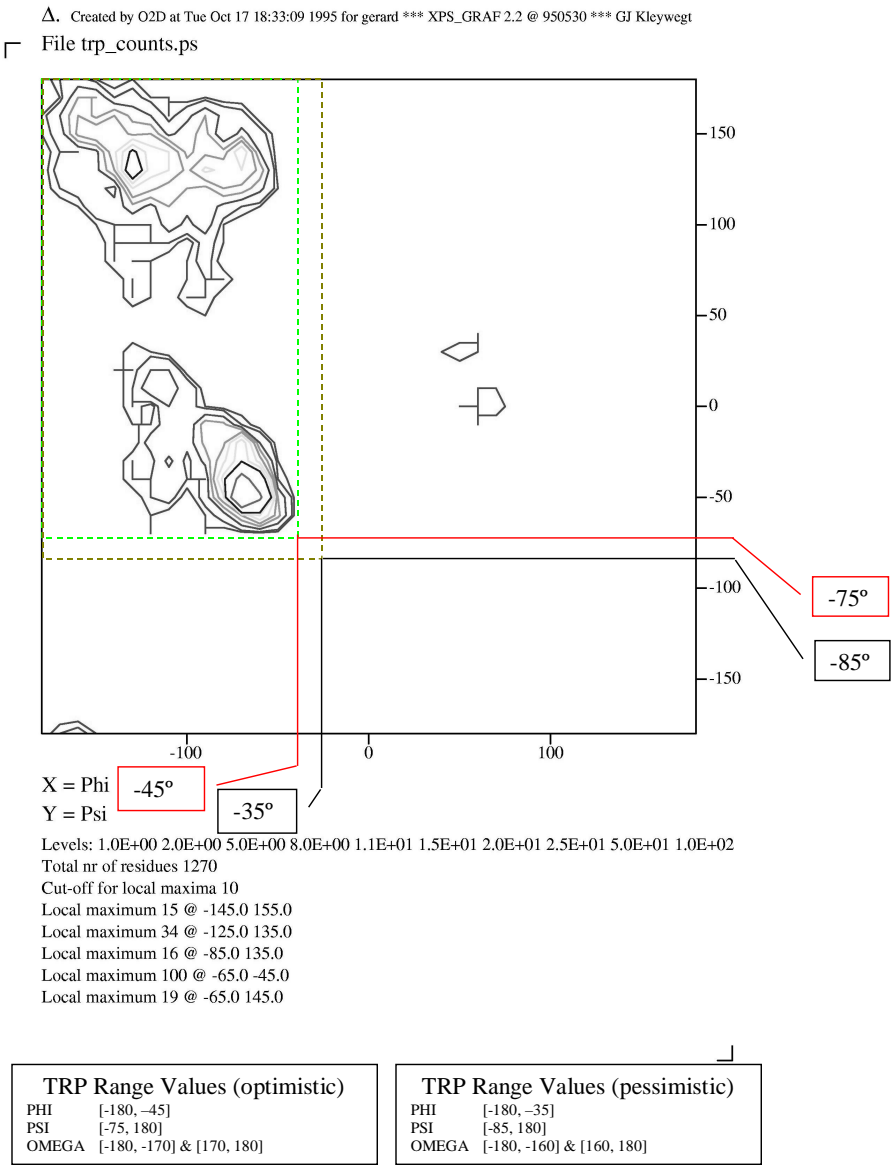


Figure C.18 Tryptophan Ramachandran Worksheet.

C.19 Tyrosine

Tyrosine, TYR, Ramachandran Worksheet

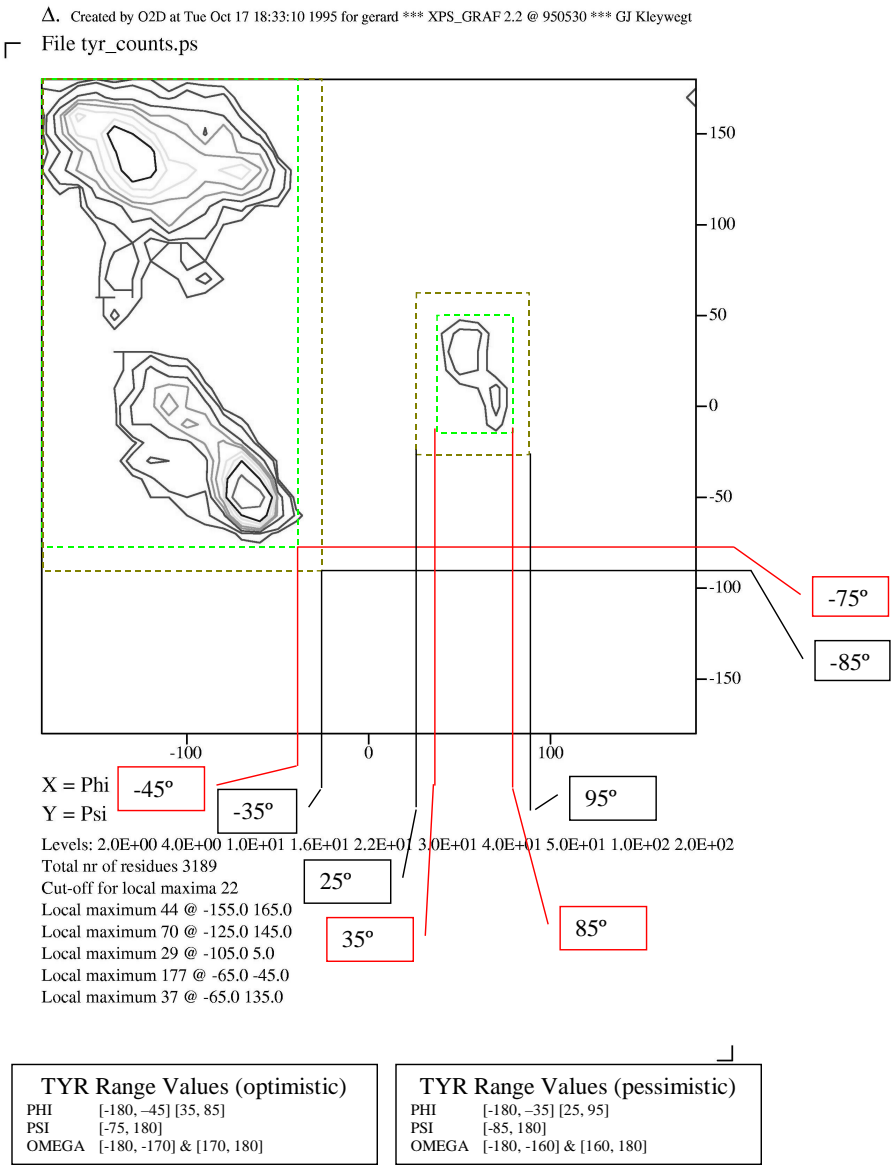


Figure C.19 Tyrosine Ramachandran Worksheet.

Valine, VAL, Ramachandran Worksheet

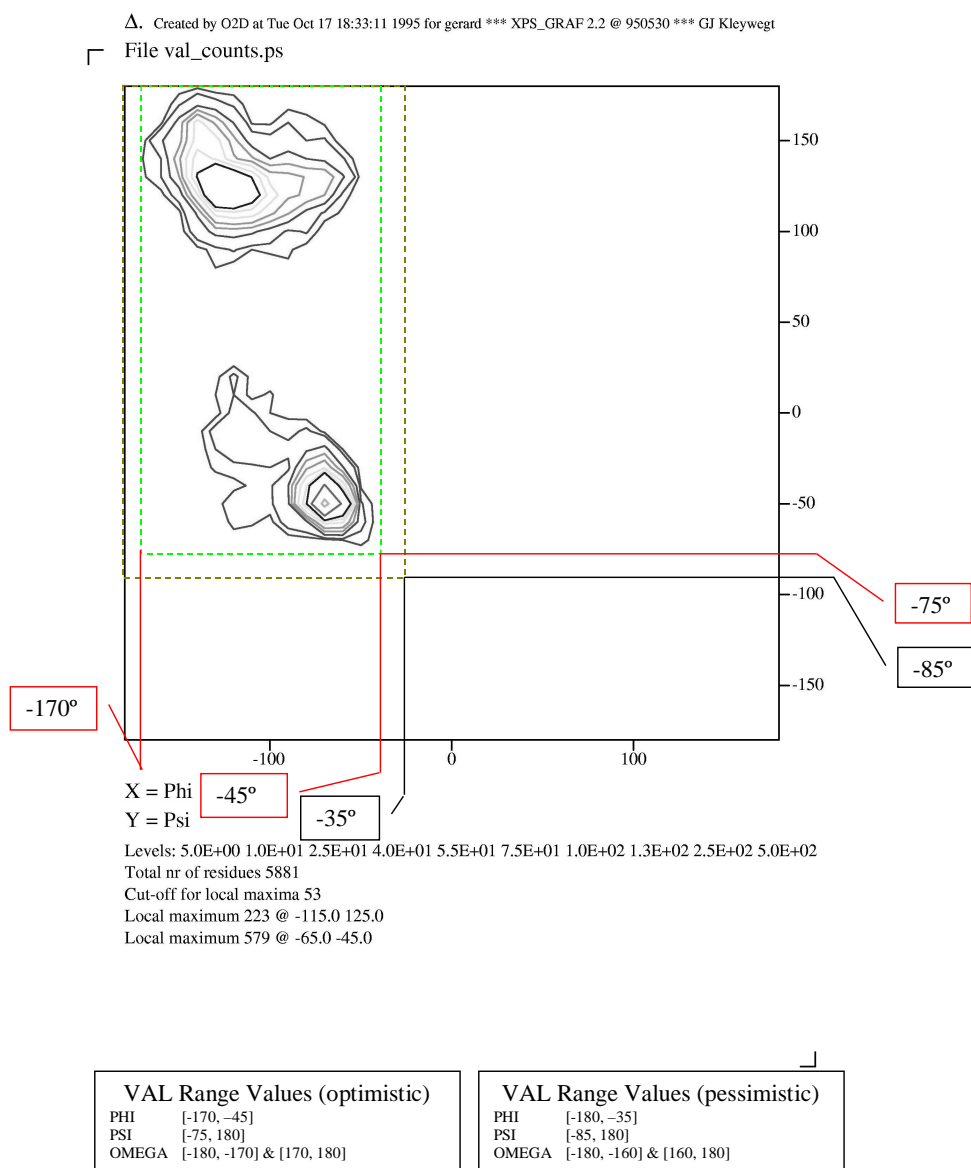


Figure C.20 Valine Ramachandran Worksheet.

Appendix D. Computational Platforms

Pile of PCs Is a heterogeneous Beowulf, running Linux 6.2 and Linux 7.1, with a 1Gbps backbone that connects two 100Mbps Ethernet sub-nets. Table D.1 describes node configurations.

D.0.0.1 PPCs' three configurations. Although the PPCs are connected as one heterogenous network with heterogenous systems, it can be decomposed into the following three different network configurations for throughput potential:

1. Entire network
2. Two matrixed Intel 510T Switches
3. Gigabit Switch

Analyzing this network would take some effort because the two patched Intel switches (one 24 port and one 12 port) are up linked to the Gigabit Switch (7 port) making the network two smaller networks connected via a 1Gbps channel. If we were to analyze these together, it would have a bisection width [63] of 1. However, for a separate analysis of these smaller networks our bisection width would be equal to the number of ports on our switches (Illustrated in

<i>Pile of PCs specifications</i>	
32 Nodes	
Systems nodes included in cluster (Number/type processor)	
8 /	1.7GHz Intel Pentium IV (P-IV) Linux 7.1 (2.4 Kernel)
2 /	1.2GHz Pentium III (P-III) Linux 7.1 (2.4 Kernel)
5 /	1.0GHz Pentium III (P-III) Linux 6.2 (2.2-14.5 Kernel)
2 /	933MHz Pentium III (P-III) Linux 6.2 (2.2-14.5 Kernel)
8 /	600MHz Pentium III (P-III) Linux 6.2 (2.2-14.5 Kernel)
4 /	450MHz Pentium III (P-III) Linux 6.2 (2.2-14.5 Kernel)
3 /	400MHz Pentium III (P-III) Linux 6.1 (2.0 Kernel)

Table D.1 PPC cluster *Specifications*

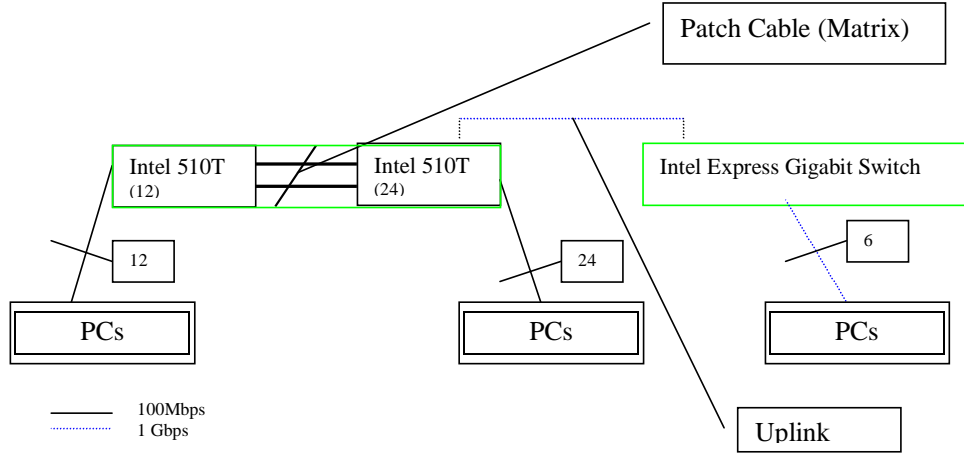


Figure D.1 An illustration of the Pile of PCs network configuration. This configuration is best described as two separate crossbar switches; furthermore, it was used as such during experimental runs in this Thesis.

Figure D.1). Making the assumption that the ports are fully utilized (This is not the case with the Pile of PCs, but it could be if we had more computers) the bisection bandwidth [63] for the entire network is 1, for just the two patched Intel switches is 36, and for the Gigabit switch is 7. The crossbar network [63] bisection calculation procedure is illustrated in Figure D.2. Channel width [63] for all three configurations is 2 because we have a transmit and receive signal for each channel. This is ignoring the fact that the channel also sends a negative signal for both transmit and receive signals at the same time to reduce emissions. Channel rates [63] run at 100Mbps for the Intel 510T switches and 1Gbps for the Gigabit Switch; however, when evaluating the entire network as a whole, one must run at the lowest denominator, which is 100Mbps in this case. Channel Bandwidth for each configuration is simply two times the Channel Rate because all channel widths were found to be two. The Bisection Bandwidth is the Bisection Width multiplied by the Channel Bandwidth, which is 200Mbps, 7.2Gbps, and 14Gbps for configurations 1, 2 and 3 respectively. Finally, the aggregate bandwidth [63] is the number of processors multiplied by Channel rate yielding 4.3Gbps, 3.6Gbps, and 7Gbps

<i>Summary of Throughput Potential</i>	
Diameter = 1	(assume the switch is not a hop)
Bisection Width = 1, 36, 7	(discussed above)
Channel Width = 2, 2, 2	
Channel Rate = 100Mbps, 100Mbps, 1Gbps	
Channel Bandwidth = 200Mbps, 200Mbps, 2Gbps	
Bisection Bandwidth = 200Mbps, 7.2Gbps, 14Gbps	
Aggregate Bandwidth = 4.3Gbps, 3.6Gbps, 7Gbps	

Table D.2 Summary of Pile of PCs (Switches combined, 2 Intel switches, 1 gigabit switch)

<i>COW Specifications</i>	
8, Sun Corporation, Ultra 10s	
Solaris 8	
Myrinet and ethernet backbone	

Table D.3 Specifications for the Cluster of Workstations

for the configurations 1,2 and 3 respectively. All parallel versions run on the PPCs used configuration 2.

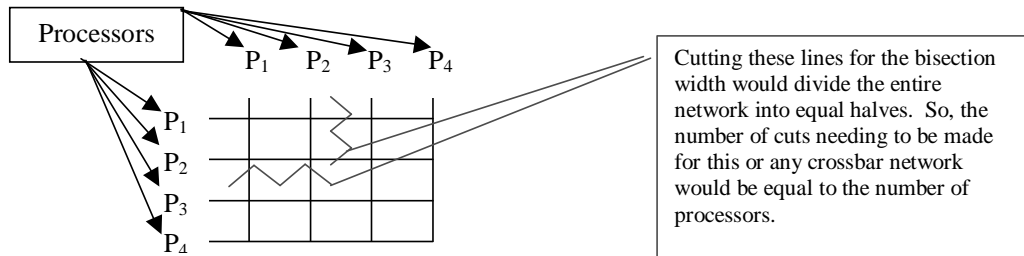


Figure D.2 An illustration of how to cut the crossbar switches when calculating bisection width.

Cluster of Workstations The cows is a homogeneous cluster of Ultra 10 Sun workstations running on a myrinet and ethernet backbone. The backbone choice is communication library configurable – identified by using myrinet IPs when communicating over the myrinet backbone and ethernet IPs when communicating over the ethernet backbone.

Diameter is 1 for this network assuming the myrinet switch is not a processor - no hops. The bisection width is 8 (See Figure D.3). The Channel width,

<i>COW Summary of Throughput Potential</i>	
Diameter = 1	
Bisection Width = 8	
Channel Width = 2(18)	
Channel Rate = 1.28Gbps	
Channel Bandwidth = 2.56Gbps	
Bisection Bandwidth = 540Gbps	
Aggregate Bandwidth = 270Gbps	

Table D.4 Summary of Throughput Potential for the Cluster of Workstations)

according to the web site is 18; however, they are assuming that there is a set of nine wires that transmits and another set of nine wires that receives, so for pedagogical purposes we set the channel width to two. The Channel Rate for those 9 wires is given as 1.28 Gbps. Furthermore, the Channel Bandwidth is twice that 2.56Gbps. The Bisection Bandwidth is the Bisection Width multiplied by the Channel Bandwidth, 20.48Gbps. Finally, the aggregate bandwidth is the number of processors multiplied by Channel rate, 10.24Gbps.

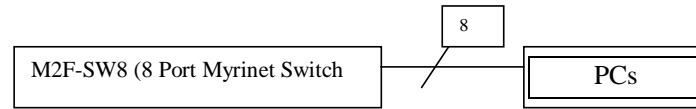


Figure D.3 An illustration of how to cut the myrnet's crossbar switches when calculating bisection width. Assuming that only one wire is cut to disconnect several processors from any one processor

Networks of Workstations Is a homogeneous Beowulf, running Linux 7.1, with a 100Mbps backbone and 100Mbps Ethernet sub-nets. The following describes node configurations:

This NOW's network summary is described in PPCs' configuration 1 summary.

SP P3 The SP P3 is an omega network with 4 CPUs per node (Illustrated in Table D.6. The network is built with 128 switching nodes - not 130. This yields $\log_2 128$ or 7 stages in the SP P3 omega network. The extra 2 nodes are used for control or management. Figure D.4 depicts our omega network.

<i>Network of Workstation Specifications</i>	
17 Node single processor nodes	
16 Compute Nodes / 1 Server Node	
AMD Processor architecture (1.4MHz)	
Work Space 3.2 GByte per Compute Node	
768 Megabyte Memory per Compute Node	
64 L1 and 256 L1 Cache per processor on Compute Nodes	

Table D.5 Specifications for the Network of Workstations)

<i>SP P3 specifications</i>
132 Nodes (4 Processors per Node)
528 Processor Elements(PEs)
130 Compute Nodes / 2 Interactive Nodes
4 Gigabyte Memory per Compute Node
2.4 Terabytes Work Space
RS/6000 Processor architecture (375 MHz)

Table D.6 SP P3 *Specifications*

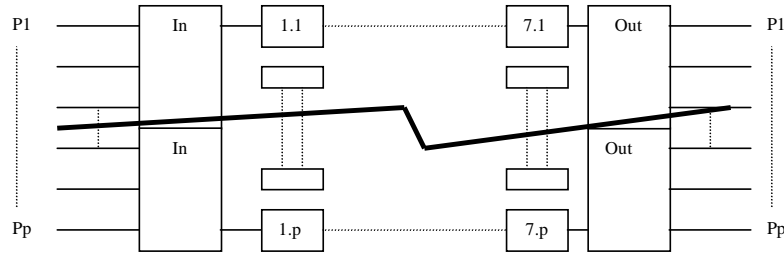


Figure D.4 An illustration of a SP P3 omega network configuration.

Assuming that the SP P3 acts as a completely connect network it can be concluded that the diameter is 1. By reconfiguring the omega network into two separate networks the *bisection width* is found by taking the difference of wires required to build the omega network before and after we separate it into two omega networks. The following was used to calculate the bisection width for the SP P3:

Removed wires = (original wires for a 128 nodes - new number of wires for two 64 node networks)

$$\text{Removed wires} = (2 * 128 * lg128) - (2 * 2 * 64 * lg64) = 256$$

<i>Summary of Throughput Potential</i>	
Diameter = 1	
Bisection Width = 256	
Channel Width = 1	
Channel Rate = 264MB/sec	
Channel Bandwidth = 264MB/sec	
Bisection Bandwidth = 540Gbps	
Aggregate Bandwidth = 270Gbps	

Table D.7 Summary of SP P3 Throughput Potential

Buses send and receive by broadcasting their message - laying signals on the bus that must travel in all directions. Thus, buses have a natural channel width of 1 unless it is a double decker bus, like the Bristol VRT3, where it can hold up to almost twice the number of passengers. The channel rate was found on the IBM web site. I used the I/O channel rate specified for the high node, which was 264MB/sec for a dual bus. Channel Bandwidth simply equals the channel rate because we have a channel width of one. The Bisection Bandwidth is the Bisection Width multiplied by the Channel Bandwidth, which comes out to be 540Gbps. Making the assumption that each of the 128 nodes is the same as a processor (node = processor), the aggregate bandwidth for the SP P3 was found to be the number of processors multiplied by Channel rate and equals 270Gbps. A summary of these values are in Table D.7

Appendix E. *fmGA Configuration File*

```
/*****  
/*****/
```

Competitive Template File

'c' in the first character of line identifies that a configuration follows

'm' in the first character of line identifies that a objective is defined on that line

The following two Formats are acceptable for a configuration

1) c s e t i

2) m # # # # #

```
/*****/  
/*****/  
/*****/
```

c - (char) identifier that this is a line adding a Competitive Template (CT)

s - (int) the number of sweeps to perform on this particular CT

e - (char) 'f' retards any evolving of the CT & 't' allows evolving to occur

t - (char) type of CT generation: 'a'=alpha, 'b'= beta, 'r'=random, 't'=test

i - (int) is there a provided CT 0=No 1=Yes

```
/*****/  
/*****/  
/*****/
```

m - (char) identifier specifying how the objectives are to be searched

- (int) There are five objectives. The objectives are grouped according to how many and which ones are specified on each line.

1) Non Bonded Energy

2) Non Bonded Energy ONE FOUR

3) Independent/dependent Bonds

4) Independent/dependent Bond Angles

5) Independent/dependent Dihedral angles

/*****

Example

m 1 2

m 3 4 5

This creates a multiobjective search with 2 objective functions

/*****

/*****

/*****

c 7 t r 0

m 1 2

m 3 4 5

/*****END OF FILE*****/

Appendix F. Biological Relations to GA Operators

F.1 Biology Background

F.1.1 Reproduction. Reproduction is the method in which life propagates itself. This occurs in two fashions, sexual and asexual. Sexual requires two gametes, one male and one female. Asexual is accomplished by one individual alone. Asexual reproduction is simpler, and is usually only accomplished by simple organisms such as bacteria and other microbes. The most common form is binary fission. In this manner, one cell divides to create a new organism. There is no exchange or shuffle of genetic material. If no mutation occurs, the offspring is an exact copy of the parent. Sexual reproduction requires the joining of two haploid gamete cells. Each parent provides $\frac{1}{2}$ of the needed genetic material. These haploid cells now having all the needed genetic material to complete a new organism may then joint to form a zygote. Sexual reproduction allows for the shuffling of gene values expressed in the offspring, although it cannot create new gene values.

F.1.2 Competition. As previously discussed, organisms reproduce themselves in their environment. Each new individual requires resources in that environment. As a population grows, these limited resources become scarce. Different members of the population are forced to compete for these scarce resources that are vital to life. Members that possess traits allowing them to gain the resource are more likely to survive.

F.1.3 Selection. As a result of competition, some individuals receive life sustaining resources while others do not. The members of the population that have the ability to more readily gain these resources tend to live longer and reproduce more. Since these members of the population tend to live longer and reproduce more, their genetic traits are passed to more offspring at a higher rate, while less fit individuals cease to pass their less fit code to their progeny. In this manner,

populations tend to fill their living space with the individuals that are most fit to survive in that space.

F.1.4 Crossover. *Meiosis* and *Mitosis* both can perform the function of cell division. Cells spend most of their life in this life function, also known as *Interphase*. Meiosis is the formation of gametes in sexual reproduction always yielding a doubling of the chromosome number. There is evidence that, in some species, proteinaceous thickening occurs within the crossbands of the synaptonemal complex at sites, which later develop chiasmata. These *recombination nodules* are thought to play a role in snipping homologous chromatids at the same site and interchanging the two resulting chromatid segments. This splicing of a length of maternal chromatid on a paternal chromatid stump and subsequent annealing of the corresponding paternal chromatid segment on a maternal stump produce the hybrid chromosomes that contribute to genetic variability. [31] Mitosis, Asexual Reproduction or Cloning, produces two daughter nuclei; each with a genetic complement identical to the parent.

F.1.5 Mutation. Mutation is a direct permanent change in a gene. This is essentially a change in the nitrogen bases that code for the gene in question. This change may be spontaneous or it may occur because the gene was exposed to a mutagen. If a mutation occurs in a somatic cell, that mutation cannot be passed to offspring. However, if the mutation occurs in a germ cell, the offspring may inherit the new gene values. This is how new allele values come into existence. While the processes of crossover and sexual reproduction may shuffle the genes possessed by offspring, mutation is the only manner in which new values for these genes may be created. There are several mutation methods that occur within an organism:

1. Point Mutation: A single point mutation, also called a base substitution, occurs when a single nucleotide is replaced with a different nucleotide. It results in base pair substitution after replication and possibly mutant proteins after transcription and translation.

- (a) Silent Mutation: This causes no change in protein generation activity. Most amino acids have multiple encodings. This being said, a mutation may change the codon in such a way that it still codes for the same amino acid. This usually occurs in third location of mRNA codon.
 - (b) Missence Mutation: The change in the codon results in a change in the amino acid coded for. This may result in harmful or beneficial protein function.
 - (c) Nonsense Mutation: The change in the codon results in an erroneous start or stop codon. This prematurely halts translation, and usually results in a non-functioning protein.
2. Frame Shift Mutation: Results from the insertion or deletion of one or more base pairs. This essentially shifts all of the codons in an "off by one" manner. Therefore all of the following codons are incorrectly coded for. This usually results in catastrophic failure of the protein.[6]

F.1.6 Transposition. This is like a cut and paste operation. A transposon has the ability to break DNA at a target site, insert itself into the target site, and then replicate the base pairings at the cut to integrate itself into the strand. These are also known as "jumping genes."

F.1.7 Translocation. This is a shift in the location of genetic code from one segment of the genome to another.

F.1.8 Conjugation. Conjugation is much like sexual reproduction. In this operation, genetic material from one cell is exchanged with genetic material from another cell. This usually occurs in simple one celled organisms such as bacteria.

F.1.9 Inversion. A chromosomal mutation involving the removal of a chromosome segment, its rotation through 180 degrees, and its reinsertion in the same location.

F.1.10 Transduction. The movement of genes from a donor to a bacterial recipient using a phage as the vector. A process whereby a cell can gain access to and incorporate foreign DNA brought in by a viral particle. Transduction usually occurs in simple one celled organisms such as bacteria.

F.1.11 Gametogenesis. The formation of gametes requiring the number of chromosomes in the gamete-forming cells to be halved. This is normally accomplished during the process of meiosis and results in the production of haploid cells.

F.1.12 Transcription. The process by which a messenger molecule is created from a DNA template. This messenger molecule is particular species of RNA called messenger RNA (mRNA).

F.1.13 Translation. After transcription (discussed above) is complete, the mRNA joins with ribosomes of the cytoplasm and other accessory molecules to synthesize a protein. This process of synthesis is called translation.

F.1.14 Exons. Exons represent message material that get translated into protein.

F.1.15 Introns. Introns are intervening stretches of DNA lying between exons. These must eventually be removed from the final mRNA product as they are not used.

Appendix G. Building Blocks

Building blocks are partial strings containing “good” information. The concept of BBs is that based on the Schema Theorem [101] and the idea that “good” solutions exist. If an analysis is completed on all of these “good” solutions, one finds a number of loci within these solutions that have the same allele values. For example, one may find that the string 1X0X, where the X’s represent don’t care bits, typically produces solutions of “better” fitness than strings of 1X1X. In this case 1X0X would be considered a “good” BB. The quality of a BB is measured through the fitness evaluation of the BB in conjunction with the competitive template, i.e. a “better” fitness value represents a “better” BB. [19]

The pfmGA revolves around the idea that a randomly created population of individuals of a specified size contain a “good” distribution of ones and zeros across the various bit positions. Since there is a “good” distribution of bits across the population members, “good” BBs also exist in the population. If these “good” BBs are found and through the genetic process of recombination are combined, the result is that the algorithm generates “good” solutions to the problem.

The building block analysis is performed in an attempt to identify the building block sizes that result in finding better solutions for Polyalanine. In this paper, a BB is a partial string representing bits from one, some, or all of the dihedral angles that each chromosome represents. The BBs are not restricted to be contiguous bits from the chromosomes but instead can be non-contiguous bits from the chromosome. Therefore if one purely looks at just one BB it may represent a whole dihedral angle or just various bits of multiple angles.

The BB analysis conducted covers a variety of BB sizes and compares the results to determine which size produces the best statistical results. One expects a BB size of 35 bits to yield the best because it is known that Polyalanine folds into an alpha helix [12] structure and Alpha helix proteins are known *a priori* to

have 3.5 residues per turn [8]. Furthermore, it can be projected that 10 meaningful bits represents dihedral angles making up one residue; therefore, 3.5 residues can be represented by 35 bits.

Appendix H. Other General AFIT Approaches

H.1 Combined Algorithms (local search)

There are many different mechanisms that can be added to any one of the aforementioned algorithms. Normally mechanism to help a GA search are memetic [78] in nature where there is a local search conducted after reaching a point in the execution of the algorithm where the algorithm has found good solutions. The following approaches have been applied to AFIT's fmGA: Lamarckian and Baldwinian (See Appendix J for explanation of these approaches). In addition to local search there can be other mechanisms applied to ensure a good spread of population members are held – this is called Niching and uses the following two approaches to ensure a good spread is maintained: Crowding and Sharing. These local search techniques employ the conjugate gradient calculation in order to find the best local solution. The conjugate gradient technique is simply taking the derivative of the energy function to find out the concavity of the solution. In doing so, further calculations can follow find out if the solution is in a local minimum or maximum, using the concavity rules of calculus. These techniques were studied by Captain Robert Gaulke [35]. The conjugate gradient using Lamarchkian and Baldwinian approaches are fmGA code options used in this thesis investigation.

The fmGA is a complicated algorithm. Increasing the complexity of AFIT's fmGA is our attempt to increase its efficiency and affectiveness. Furthermore, AFIT's fmGA has numerous settings and mechanisms that are discussed in detailed in Chapters 3, 4 and 5. Following Captain Gualke's memetic approaches in 1999, Captain Steven R. Michaud also utilized the fmGA in attempts to identify good building blocks that matched secondary structure patterns. His research followed the NX (INTEL) version of the fmGA code which was dated; however, semi-effective.

H.2 Hybridized GA

Captain Charles E. Kaiser followed after Captain Gaulke's work by using Hybridized GA approached to solving the PSP problem in 1995. Kaiser's work does not fit the usual *hybridized* approaches where the attempt is to use a GA tool in conjunction with heuristic solvers or other mechanisms. Furthermore, the hybrid GA used in the case of Kaiser's thesis is simply solutions gained by a GA supported by his own analytical skills. [1] discusses the appropriateness and need for a heuristic solver to a GA approach – making Kaiser's work [50] important; however, better solutions weren't discovered. In addition to Kaiser's hybrid model, he also studied a "farming model" and "island model" parallel GA (PHGA) used to increase efficiency of his hybrid GA.

H.3 Real Valued GA

In addition to Kaiser's work with hybridized GAs, he also attempted to solve the PSP problem with a real valued GA and found better results than a competitor (Scheraga, et al) [50]; however, today we know that neither method produced superior solutions. This approach was innovative to the PSP problem; however, there are many other implementations of the real valued GA [73]. In fact, Kaiser integrated Michalewicz's real valued GA into the protein model and CHARMM fitness function. A real valued GA imposes real numbers as a replacement for the binary strings used within the fmGA. Real valued GAs have solution granularity advantages over binary GAs. For instance, a GA using binary numbers have limited ranges. They can only take on the following values: 0, 2^1 , 2^2 , \dots , and 2^n – where n is the number of bits available for each number. Whereas, real valued GAs have only the limits of the computer hardware available to represent each number. For the PSP problem, this kind of advantage could be the difference in getting a semi-optimal solution and getting the true solution. Moreover, if your GA cannot represent the answer because the problem has been oversimplified, the GA never finds the optimal solution.

H.4 Linkage Learning GA

The Linkage Learning Genetic Algorithm was designed by David Goldberg's research group to "solve problems of bounded difficulty quickly, reliably, and accurately" [43]. In an attempt to mimic the linkage between the DNA to Protein mapping, this algorithm hunts for advantages in "tight linkage" exploitation. Furthermore, it applies a new two-point crossover operator to a new chromosome mapping [20]. This operator uses the idea of Transposition¹ for reproduction by grafting a chromosome into a recipient at a random location. Overall the algorithm works similarly to the sGA; however, where the sGA failed, this algorithm picks up the lost linkages. These lost linkages are important to finding a path from suboptimal solutions to semi-optimal solutions.

¹This is like a cut and paste operation. A transposition has the ability to break DNA at a target site, insert itself into the target site, and then replicate the base pairings at the cut to integrate itself into the strand. These are also known as "jumping genes."

Appendix I. Basic Evolutionary Algorithmic Approaches Justified

I.1 Evolutionary Strategies

Evolutionary Strategies originated in Germany where Bienert, Rechenberg and Schwefel applied it to optimizing the drag on a pipe or nozzle. They achieved good semi-optimal solutions with the application of ES. Their simple ES had merely one population member where they applied mutation to find more optimal solutions.

I.1.1 Mutation. Moreover, this first ES used a simple two membered, one n-dimensional, real-valued vector of object variables which is mutated by applying identical standard deviation to each variable. This straight forward local search mechanism can be effective on the right problem; however, for the PSP problem, I suggest the application of a more state-of-the-art ES that has a self-adapting mechanism optimizing both the parameter and objective variables. The self-adapting ES uses autocorrelation functions to allow for the mutation area to be stretched in directions that are mathematically suspected to include areas of the fitness landscape which might have better solutions. This is emphnot to say that, because the covariance and standard deviation has adapted to allow for the finding of better solutions, the algorithm must find better solutions. It still is a stochastic mutation function that attempts to condensate for the areas where better answers are thought (statistically) to be.

I.1.2 Recombination. This mechanism has two operators used to accomplish recombination. One is sexual and the other is panmictic. The first is a sexual operator that randomly selects two parents each time and creates one offspring. The second is a panmictic operator meaning the first parent is selected and then held on to for mating with many other parents.

I.1.3 Selection. Selection within ES are deterministic. Normally a $(\mu + \lambda)$ would be the selection mechanism used; however this has been shown to slow the self-adaptation mechanism with respect to the strategy parameter. Therefore, (μ, λ) is the selection mechanism used in solving the PSP problem.

I.1.4 Application. Application of ES to the PSP problem is kind of straight forward. The encoding of the protein from the phenotype (protein) to the genotype ending with the entire protein specified, atom by atom, in groups within dihedral angles. These dihedral angles are then ladled to hold specified places as real numbers in the chromosome.

Evolutionary Strategy Algorithm

$t = 0$

Initialize population, $P(0), = \{\vec{d}_1, \vec{d}_2, \dots, \vec{d}_\mu\} \in I^\mu$

d_i : are filtered with a feasibility function using constraints

Evaluate $P(0)$:where each member of the population is evaluated

while (Generations are not satisfied) do

 recombine

 mutate

 evaluate

 select

$t = t + 1$

od

It should be noted that the distributions used for the mutations should take into account that they can go to a maximum of 360 degrees – and even less if the Ramachadrin Plots are applied. Therefore, the distribution needed might be that of a gama distribution.

I.2 Evolutionary Programming

Evolutionary Programming (EP) is similar to that of ES. Mutation is normally distributed and have some self-adaptation scheduled into genotype mutations. Again, a state-of-the-art implementation of this EP called *meta*-EP should be used in solving the PSP problem.

I.2.1 Mutation. The asexual mutation operator mutates the population member with a standard deviation that is obtained for each component (dihedral angle) of the object variable vector as the square root of a linear transformation of the fitness function. In overcoming tuning problems as the algorithm runs, they have added a vector of variances per individual. This vector is very much similar to the parameter variables for the ES.

I.2.2 Recombination. EP does not use crossover or recombination, but relies heavily upon the mutation operator discussed above.

I.2.3 Selection. The asexual essence of the mutation operator the offspring become the size of the population. Additionally, tournament selection is applied with ranking (in descending order).

I.2.4 Application. Considering this method is so similar to that of ES, it can be applied to the PSP problem in a similar way as well. The encoding of the protein into dihedral angles is applied again and then the real values are used within the chromosomes. The algorithm looks the same as the ES algorithm, but you remove the recombination. There is nothing special and it should be that Evolutionary Programming should be a sub class of the Evolutionary Strategies.

I.3 Genetic Algorithms (GA)

I.3.1 fast messy Genetic Algorithm (fmGA). The first step to applying a GA is to transform the problem domain into a fixed length binary string - called chromosomes. In other words, a solution should be representable by one chromosome. Individual elements of a chromosome are called features - corresponding to the genes of a chromosome. Feature values are the values that one feature may take on - these represent alleles of a gene. Finally, the set of every allele is the genetic alphabet [2]. After a discretized encoding scheme is applied to the problem, there must be a way to decode and evaluate the merit of a specific chromosome, or solution. This is normally called the fitness function - it checks the fitness or merit of a solution. Its main purpose is to give an indicator if one chromosome is better than another. Unfortunately, fitness evaluation causes a decode to occur and a high computational analysis of the chromosome usually costing the algorithm in time (Such as the fitness function in our GA used to search the PSP problem energy landscape).

The main routine in a GA, after encoding the problem, builds a population of chromosomes. It then selects from the current population and uses reproduction, crossover and mutation to build new population members - each time evaluating the newly created chromosome's fitness. Upon evaluation of a better chromosome, that better chromosome is placed into the population. The routine then repeats itself. Figure 3.7 illustrates this cycle. The dotted lines indicate the barrier between the real solution or problem domain and the encoded solution or chromosome domain. This GA can also be referred to as a steady state GA where the population size remains the same throughout the execution of the program.

I.3.2 Application.

fast messy Genetic Algorithm

Stochastic-Search-GA Algorithm Specifications Extended Iterative Form

Step(0) Initialization Randomly generate a solution P_i

Set number Max tries

Set t tries to zero

Set Best to highest number possible

Step(1)

Generator Next State using Population Pop

Step(2)

If feasible(P_t) then add to Pop

If $f(P_t) < f(\text{Best})$ then $\text{Best} = P_t$

Step(3)

If Max tries $> t$ then Stop

Goto Step 1

I.4 Genetic Programming

It is most challenging to get a computer to accomplish a task without coaching it in how to achieve that task. Genetic Programming (GP) embodies this concept of a computer learning how to program itself, or auto programming. GP does this by genetically developing or allowing a population to evolve using Darwinian's *natural selection* along with biologically understood operations. These operations include, but are not held exclusively to the following: reproduction, crossover, mutation, and architecture-altering operations patterned after gene duplication and deletion.

I.4.1 Description.

I.4.1.1 Population Creation. The auto programming begins by randomly generation of many computer programs. These computer programs may be in the form of mathematical logic (represented by reverse polish) or in the form of program modules or sections. In the case of applying GP to the PSP problem, the

modules could be in the form of localized search operators or different GAs altogether. So we begin by massively producing combinations of GAs that can work on a similar problem. We can use differing GAs (simple GA, messy GA, fast messy GA, etc) and operators (conjugate gradient, local twist, sweep, etc) to swap out after a conclusive run on a particular configurations. I know a picture would be nice here, but I can't get latex to play nice. Basically, for terminals you may have an entire GA or just an operator. A higher level GP can manage each configuration and keep track of what configurations gave the best result. The population are different for each differing configurations generated by the operators in GP. The fitness evaluation of each population member are time intensive, but eventually it must yield a result. To run an experiment of this proportion, one must be committed to run it for a year or so on today's HPCs.

I.4.1.2 Reproduction. This operator simply selects the next generation based on fitness values. Once all population member have been evaluated and other operations have been applied – configurations finding the best fitness (lowest energy) are moved to the next population pool.

I.4.1.3 Crossover. This operator is sexual reproduction after the selection of two parental programs. In our case, this would be the selection of two configurations. Crossover points are then randomly chosen in each parent. The subtree at the crossover point for the first parent is deleted and replaced with that at the second parent's crossover point subtree.

I.4.1.4 Mutation. The mutation operator selects a configuration to mutate based on fitness. Randomly selects a mutate point, deletes the subtree at that point, and grows another subtree at the mutation point to replace it.

I.4.1.5 Architecture-Altering Operations. This operator dynamically allows the removal and insertion of sub-routines within programs genetically. This

is to allow for the actual program to be modified or shaped in a way so that it may adjust to solving the problem. Ultimately, this operator relieves the programmer from writing engrained specifications for a program. This operator would be at the heart of how I'm suggesting the PSP problem to be solved using GPs. This operator would mutate sub-routines that are already being 'wholly' crossed and mutated in and out of population members.

Appendix J. Data and task decomposition design

J.1 Multi Objective fmGA Data and Task decomposition

Data and task decomposition for the MOfmGA selected can be simple or complex depending on the design. The fmGA used to solve the PSP problem at AFIT can be made to be either single program single data (SPSD) or single program multiple data (SPMD). With the addition of farms (new mechanism using fine granularity parallelism), the program has now become able to be multiple program multiple data (MPMD). It is interesting to have the different models for several reasons. When the GA runs using the single data model, the GAs running in parallel are generating populations separately from one another and only interact when a migration of a good population occurs with some probability. This is a good model to use when communication cost is high. Furthermore, if you had a shared memory machine, communication does not need to occur it is more advantageous to use a MPMD setup where data does not have to be transferred and data pipelining can be utilized (requirement of control parallelism) - see the description of the farming model of the fmGA.

There are many design issues with making a GA run in parallel: Level at which you want to parallelize the GA (GA itself, operators, fitness evaluation, population pool), distributed system type (multi-computer or multi-processor), memory setup (shared or distributed), network interconnectivity (Mesh, Hypercube, or Ring), and inter-process communication setup between nodes or processors to name a few. Von Neumann-based parallel processing systems are categorized accordingly: Multiple Instruction Multiple Data (MIMD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD), or Single Instruction Single Data (SISD) paradigm [63]. Additionally, there are special cases under categories. For example, MIMD can be Single Program Multiple Data (SPMD) or Multiple Program Multiple Data (MPMD). MIMD normally consists of several different processors capable of

running different instructions on different data sets independently. AFIT has this type of setup, two multi-computer clusters called the Pile of PCs and COWs. In addition to the systems at AFIT, we have available at our disposal the SP-3 at the MSRC. The SP-3 is a multi-processor machine with shared memory and many processors. So, there is a need for a parallel implementation of this GA. Currently there are 3 brands of the parallel version of the fmGA used at AFIT:

1. Independent: The normal implementation of the fmGA or MGA using a single node and processor.
2. Combined: each processor sends best-found building blocks to the master processors at the end of each juxtaposition phase.
3. Global combine: each processor exchanges their best building blocks with all other processors. This is called Global exchange. After that, every processor executes an independent juxtaposition phase on a copy of global population.

Furthermore, there are three models that typify how future designs of PGAs would function [9]. They are as follows:

1. Island Mode (Neither Task nor Data Decomposition) This being the simplest model where the population is divided into subpopulations that are distributed among the different processors. Evolution is conducted in parallel at each processor independent from all other processors; however, at certain intervals migration occurs between processors. Migration is the passing of solutions between processors. This model should provide near linear speedup [13] [62]. Normally, the Island model is used on coarse grained or MIMD architectures [63]. This model is a good candidate for running the PSP problem using our fmGA. Furthermore, it is our 3rd brand of the fmGA at AFIT - called Global Combine.
2. Neighborhood Model (Data Decomposition) This model splits the population up spatially into a two-dimensional or three-dimensional grid. Each string is

placed on an individual processor, forcing the crossover and selection operators to be redesigned to allow for operations across multiple processors. Normally, this is implemented on fine-grained or SIMD architectures [63]. This model is our 2nd brand of the fmGA at AFIT - called Combined.

3. Farming Model (Task Decomposition) Farming here is used in the context of manufacturing, or to farm out work [25]. In this model there is a Boss, or head node, and workers. The head node essentially farms out the work to each worker. The head node is responsible for keeping track of idle workers and load balancing schemes. This model is exploited with the new mechanism to have farm evaluate the fitness of chromosomes built from within the cut and splice process during the juxtaposition phase.

J.1.1 Design of operators and parameter values. There are many different mechanisms that can be added to any one of the aforementioned algorithms. Normally mechanism to help the search are memetic in nature where there is a local search accomplished after reaching a point in the execution of the algorithm where the algorithm has found whole solutions. The following approaches have been applied to AFIT's fmGA: Lamarckian and Baldwinian. In addition to local search there can be other mechanisms applied to ensure a good spread of population members are held - this is called Niching and uses the following two approaches to ensure a good spread is maintained: Crowding and Sharing. These local search techniques employ the conjugate gradient calculation in order to find the best local solution. The conjugate gradient technique is simply taking the derivate of the energy function to find out the concavity of the solution. In doing so, further calculations can follow find out if the solution is in a local minimum or maximum, using the concavity rules of calculus.

1. Lamarckian: The Lamarckian evolution technique utilizes a local search for improving a current population member. In addition, it also places this new-

found optimal fitness and string into the next generation. This can be handy. After selection has been performed, in the fmGA, an additional local search could be performed to make sure that the local best is obtained and passed onto the next generation. [20]

2. Baldwinian: The Baldwinian technique applies the combination of learning plus evolution. Thus, by teaching the next generation a new skill it can pass down to the next generation; likewise, the next generation does not automatically get the skill (not born with it because the previous generation has learned it). Essentially, this is applied to the fmGA by conduction a local search and finding a new optimal fitness associated with a particular string. The new lower fitness value is then updated for that string, but the string evaluates to this new lower fitness value is not replacing the starting string (meaning that the locally searched minimum is at a lower fitness value than the actual string that has taken on this new lower fitness value). [35]
3. Niching [35] Niching stems from nature where different species then to exploit separate niches (sets of environmental features) that other organisms have little or no interest rather than competing directly for the same resource. The basic idea behind this technique is that it is ill advised to have all population members having nearly the same value; otherwise, the algorithm gets stuck in some local minimum. Two niching techniques are crowding and sharing.

Crowding: Crowding is an operator that keeps track of string patterns, replacing any string that overlaps another population member. By replacing these similar strings, diversity and exploration is enhanced as well as allowing more species to evolve.

Sharing: Sharing is reducing duplicate member fitness based on how close a member is to other members. This can be tracked and employed using the hamming code distance function to determine how close strings are to one another.

J.1.2 Implementation. The implementation of the MOfmGA is a modified version of the already working fmGA. This code is used mostly because the CHARMm structs and variables are already integrated into the algorithm. Additionally, reuse of code is the best way to do research because it allows the researcher to focus on the variables entered into the algorithm, not the writing of the code. Moreover, writing code is not considered research at all. Unfortunately, the integration of the multiple objective structs into a single objective code is a significant change and has proved to be quite intense.

Appendix K. Multiobjective Discussion

K.1 Multiobjective Optimization

The process of finding the global maximum or minimum of any function is referred to as Global Optimization. In general, this is presented in Definition 1 as stated in Bäck [4]:

Definition 1 (Global Minimum): *Given a function $f : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}$, $\Omega \neq \emptyset$, for $\vec{x} \in \Omega$ the value $f^* \triangleq f(\vec{x}^*) > -\infty$ is called a global minimum if and only if*

$$\forall \vec{x} \in \Omega : f(\vec{x}^*) \leq f(\vec{x}) . \quad (\text{K.1})$$

Then, \vec{x}^ is the global minimum solution(s), f is the objective function, and the set Ω is the feasible region. The problem of determining the global minimum solution(s) is called the global optimization problem.* \square

This formulation must be modified to reflect the nature of multiobjective problems where there may not be one unique solution but a set of solutions found through the analysis of associated Pareto Optimality Theory. Many times multiobjective problems force the decision maker to make a choice which is essentially a tradeoff of one solution over another in objective space. Before we present the associated multiobjective definition, we must define what a MOP is. Multiobjective problems are those where the goal is to optimize n objective functions simultaneously. This may involve the maximization of all n functions, the minimization of all n functions or a combination of maximization and minimization of these n functions. A MOP and a MOP global minimum (or maximum) is formally defined by Van Veldhuizen as [97]:

Definition 2 (General MOP): *In general, a MOP minimizes (or maximizes) $F(\vec{x}) = (f_1(\vec{x}), \dots, f_k(\vec{x}))$ subject to $g_i(\vec{x}) \leq 0$, $i = 1, \dots, m$, $\vec{x} \in \Omega$. An MOP*

solution minimizes the components of a vector $F(\vec{x})$ where \vec{x} is a n -dimensional decision variable vector ($\vec{x} = x_1, \dots, x_n$) from some universe Ω . \square

Definition 3 (MOP Global Minimum): Given a function $F : \Omega \subseteq \mathbb{R}^n \rightarrow \mathbb{R}^k$, $\Omega \neq \emptyset$, $k \geq 2$, for $\vec{x} \in \Omega$ the set $\mathcal{PF}^* \triangleq F(\vec{x}_i^*) > (-\infty, \dots, -\infty)$ is called the global minimum if and only if

$$\forall \vec{x} \in \Omega : F(\vec{x}_i^*) \preceq F(\vec{x}) . \quad (\text{K.2})$$

Then, \vec{x}_i^* , $i = 1, \dots, n$ is the global minimum solution set (i.e., \mathcal{P}^*), F is the multiple objective function, and the set Ω is the feasible region. The problem of determining the global minimum solution set is called the MOP global optimization problem. \square

This MOP consists of k objectives reflected in the k objective functions, m constraints on the objective functions and n decision variables. The k objective functions may be linear or nonlinear in nature. The evaluation function, $F : \Omega \rightarrow \Lambda$, is a mapping from the decision variables ($\vec{x} = x_1, \dots, x_n$) to output vectors ($\vec{y} = a_1, \dots, a_k$) [97].

MOPs typically consist of competing objective functions, which may be independent or dependent on each other. An example of this is a company's quest to purchase a backbone for their computer network that provides the greatest throughput at the least monetary cost. These objectives are highly dependent on each other as increased cost brings increased throughput and vice-versa.

It is necessary to define additional terminology to remain consistent with the terminology used in the EA field. The term *objective* is used to refer to the goal of the MOP to be achieved and *objective space* is used to refer to the coordinate space within which vectors resulting from the MOP evaluation are plotted [97].

K.1.1 Pareto Terminology. The concept of Pareto Optimality is integral to the theory and analysis of MOPs. A way to determine if one solution is “better” than another is a necessity here as well as in all problems. Pareto concepts allow for

the determination of a set of optimal solutions in MOPs. Although single-objective optimization problems may have a unique optimal solution, MOPs usually have a possibly uncountable set of solutions, which when evaluated produce vectors whose components represent trade-offs in decision space. Some key Pareto concepts, for minimization MOPs, are defined mathematically by Van Veldhuizen as [97]:

Definition 4 (Pareto Dominance): *A vector $\vec{u} = (u_1, \dots, u_k)$ is said to dominate another vector $\vec{v} = (v_1, \dots, v_k)$ (denoted by $\vec{u} \preceq \vec{v}$) if and only if u is partially less than v , i.e., $\forall i \in \{1, \dots, k\}, u_i \leq v_i \wedge \exists i \in \{1, \dots, k\} : u_i < v_i$. \square*

Definition 5 (Pareto Optimality): *A solution $x \in \Omega$ is said to be Pareto optimal with respect to Ω if and only if there is no $x' \in \Omega$ for which $\vec{v} = F(x') = (f_1(x'), \dots, f_k(x'))$ dominates $\vec{u} = F(x) = (f_1(x), \dots, f_k(x))$. The phrase “Pareto optimal” is taken to mean with respect to the entire decision variable space unless otherwise specified. \square*

Definition 6 (Pareto Optimal Set): *For a given MOP $F(x)$, the Pareto optimal set (\mathcal{P}^*) is defined as:*

$$\mathcal{P}^* := \{x \in \Omega \mid \neg \exists x' \in \Omega \ F(x') \preceq F(x)\}. \quad (\text{K.3})$$

\square

Definition 7 (Pareto Front): *For a given MOP $F(x)$ and Pareto optimal set \mathcal{P}^* , the Pareto front (\mathcal{PF}^*) is defined as:*

$$\mathcal{PF}^* := \{\vec{u} = F(x) = (f_1(x), \dots, f_k(x)) \mid x \in \mathcal{P}^*\}. \quad (\text{K.4})$$

\square

Pareto optimal solutions are those solutions within the search space whose corresponding objective vector components cannot be all simultaneously improved.

These solutions are also termed *non-inferior*, *admissible*, or *efficient* solutions, with the entire set represented by \mathcal{P}^* . Their corresponding vectors are termed *nondominated*; selecting a vector(s) from this vector set (the Pareto Front set \mathcal{PF}^*) implicitly indicates acceptable Pareto optimal solutions (**genotypes**). These solutions may have no clearly apparent relationship besides their membership in the Pareto optimal set. It is simply the set of all solutions whose associated vectors are non-dominated; it is stressed here that these solutions are classified as such based on their *phenotypical* expression. Their expression (the nondominated vectors), when plotted in criterion (**phenotype**) space, is known as the *Pareto front* [97, 104].

A MOEA’s complex structure can lead to confusion in discussing the algorithmic process that takes place. To prevent further inconsistencies in discussions of MOEAs, Van Veldhuizen [97] developed Pareto terminology to clarify MOEA discussions. He stated at any given generation of a MOEA a “current” set of Pareto optimal solutions (with respect to the *current* MOEA generational population) exists and is termed $P_{current}(t)$, where t represents the generation number. There are also a number of MOEAs that use a secondary population, also referred to as an archive or an external archive, to store nondominated solutions found through the generations [98, 97]. Since this secondary population contains Pareto optimal solutions generated at a certain point in time, each time another point is considered for addition to the secondary population, the point must be looked at for non-dominance with respect to the points currently in the secondary population. This secondary population is denoted $P_{known}(t)$. The t reflects the potential changes to the secondary population as the MOEA executes. Additionally, $P_{known}(0)$ is defined as the empty set (\emptyset) and P_{known} alone as the *final* set of Pareto optimal solutions returned by the MOEA at termination [97, 104].

Different secondary population storage strategies exist; the simplest is when $P_{current}(t)$ is added at each generation (i.e., $P_{current}(t) \cup P_{known}(t-1)$). At any given time, $P_{known}(t)$ is thus the set of Pareto optimal solutions *yet found by the MOEA*

through generation t . Of course, the *true* Pareto optimal solution set (termed P_{true}) is not explicitly known for problems of any difficulty. P_{true} is defined by the functions composing an MOP; it is fixed and does not change. Because of the manner in which Pareto optimality is defined $P_{current}(t)$ is always a non-empty solution set [97].

$P_{current}(t)$, P_{known} , and P_{true} are sets of MOEA genotypes where each set's phenotypes form a Pareto front. We term the associated Pareto front for each of these solution sets as $PF_{current}(t)$, PF_{known} , and PF_{true} . Thus, when using an MOEA to solve MOPs, the implicit assumption is that one of the following holds: $P_{known} = P_{true}$, $P_{known} \subset P_{true}$, or $PF_{known} \in [PF_{true}, PF_{true} + \epsilon]$ over some norm (Euclidean, RMS, etc.).

Solutions on the Pareto Front represent optimal solutions in the sense that improving the value in one dimension of the objective function vector leads to a degradation in at least one other dimension of the objective function vector. This forces the decision maker to make a tradeoff decision when presented with a number of optimal solutions for the MOP at hand, i.e. the Pareto Front. There exists a difference in terminology between an acceptable compromise solution and a Pareto Optimal Solution [28]. The decision maker typically chooses only one of the associated Pareto Optimal solutions, $\vec{u} \in \mathcal{PF}^*$, as being the acceptable compromise solution, even though all of the Pareto Optimal solutions are optimal. The decision maker bases this solution choice off of which solutions take into account the human's preference. The human preference factor forces engineers and scientists to attempt to find all of the points on the Pareto Front since all points are not weighted equally in the decision maker's mind.

Appendix L. Data from multiple competitive template experiments on

MET

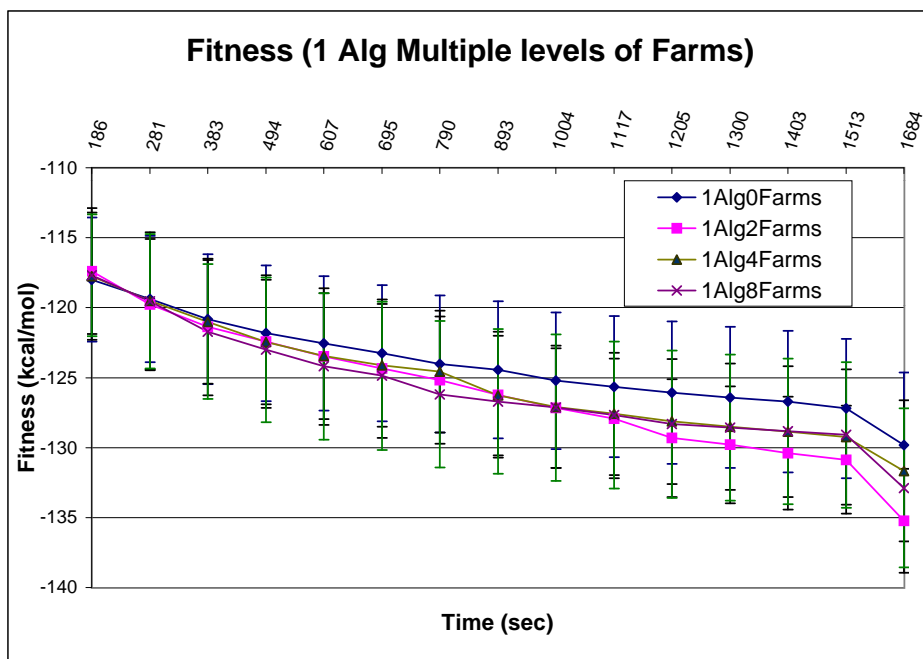
MET Data for Multiple Competitive Template Experiment							
(Group)	Exper	BB	(A)Alpha	(B)Random	(C)Beta	(D)ARB	(E)ARBpan
A	1	6	-28.278737	-22.889378	-24.905582	-28.291876	-28.300307
A	1	7	-28.309026	-22.97107	-24.928189	-28.3079	-28.308393
A	1	8	-28.342692	-23.034342	-24.943951	-28.34466	-28.32919
A	1	9	-28.360435	-23.108851	-24.951829	-28.35556	-28.364017
A	1	10	-28.367423	-23.134461	-24.96031	-28.362285	-28.397825
A	1	6	-28.379291	-23.190566	-24.989816	-28.370676	-28.411875
A	1	7	-28.385696	-23.23941	-25.381038	-28.384995	-28.414762
A	1	8	-28.411863	-23.269299	-25.661887	-28.413582	-28.425261
A	1	9	-28.433324	-23.329624	-25.673459	-28.429047	-28.441122
A	1	10	-28.470366	-24.28722	-25.682582	-28.446335	-28.471737
A	1	6	-28.492738	-25.148008	-25.704999	-28.458927	-28.484505
A	1	7	-28.506666	-25.630141	-25.734522	-28.469983	-28.492895
A	1	8	-28.523697	-26.201578	-25.743283	-28.477254	-28.500351
A	1	9	-28.530769	-26.487595	-25.753653	-28.480844	-28.508845
A	1	10	-28.68424	-29.499787	-25.838175	-28.912087	-29.524363
A	2	6	-28.416712	-5.841503	-26.926519	-28.472992	-26.926107
A	2	7	-28.50404	-7.562974	-26.973734	-28.528579	-27.055564
A	2	8	-28.759294	-8.790183	-27.029423	-28.625501	-27.118186
A	2	9	-28.871578	-12.098284	-27.130081	-28.662987	-27.201811
A	2	10	-28.912501	-20.117619	-27.162811	-28.688526	-27.38164
A	2	6	-28.947007	-21.732841	-27.307008	-28.712064	-27.451884
A	2	7	-28.983896	-22.771764	-27.445468	-28.731836	-27.548983
A	2	8	-29.11341	-23.106475	-27.517872	-28.758657	-27.58493
A	2	9	-29.254266	-23.862013	-27.548043	-28.771714	-27.772729
A	2	10	-29.347353	-24.429999	-27.560085	-28.802399	-28.078183
A	2	6	-29.388809	-24.648894	-27.569261	-28.821688	-28.120457
A	2	7	-29.404285	-24.812491	-27.585248	-28.865759	-28.150927
A	2	8	-29.424638	-24.989525	-27.5971	-28.914347	-28.18669
A	2	9	-29.452357	-25.087235	-27.61398	-28.963739	-28.205493
A	2	10	-30.023546	-26.3358	-27.727388	-30.031059	-29.005247
A	3	6	79.000315	-25.38384	-26.691103	-27.437654	-24.078155
A	3	7	29.476581	-25.496948	-26.70405	-27.492608	-24.353812
A	3	8	22.08554	-25.532022	-26.716553	-27.590383	-24.654618
A	3	9	10.057835	-25.574089	-26.843654	-27.651587	-24.816164
A	3	10	8.485627	-25.636066	-26.849769	-27.757783	-24.92522
A	3	6	7.520392	-25.820151	-27.062184	-27.797426	-25.074039
A	3	7	4.351143	-25.863905	-27.173046	-27.825934	-27.262767
A	3	8	1.080246	-25.895067	-27.257909	-27.864593	-27.670463
A	3	9	-2.587026	-25.942412	-27.339293	-27.963511	-27.986986
A	3	10	-6.85195	-26.015832	-27.392019	-28.069859	-28.354818
A	3	6	-8.394699	-26.038136	-27.414542	-28.169492	-28.639541
A	3	7	-9.416877	-26.070735	-27.424831	-28.198846	-28.697869
A	3	8	-10.16397	-26.085267	-27.441537	-28.231767	-28.784343
A	3	9	-11.523245	-26.114748	-27.454024	-28.26861	-28.812674
A	3	10	-31.460474	-31.551802	-27.888819	-31.725017	-29.739632
A	4	6	61377.21149	-20.440169	-28.65729	-29.857544	-28.402071
A	4	7	61376.48557	-21.606168	-28.746781	-29.876073	-28.410378
A	4	8	61375.46262	-22.142392	-28.798769	-29.891933	-28.42032
A	4	9	61374.72659	-22.368954	-28.878908	-29.918086	-28.458648

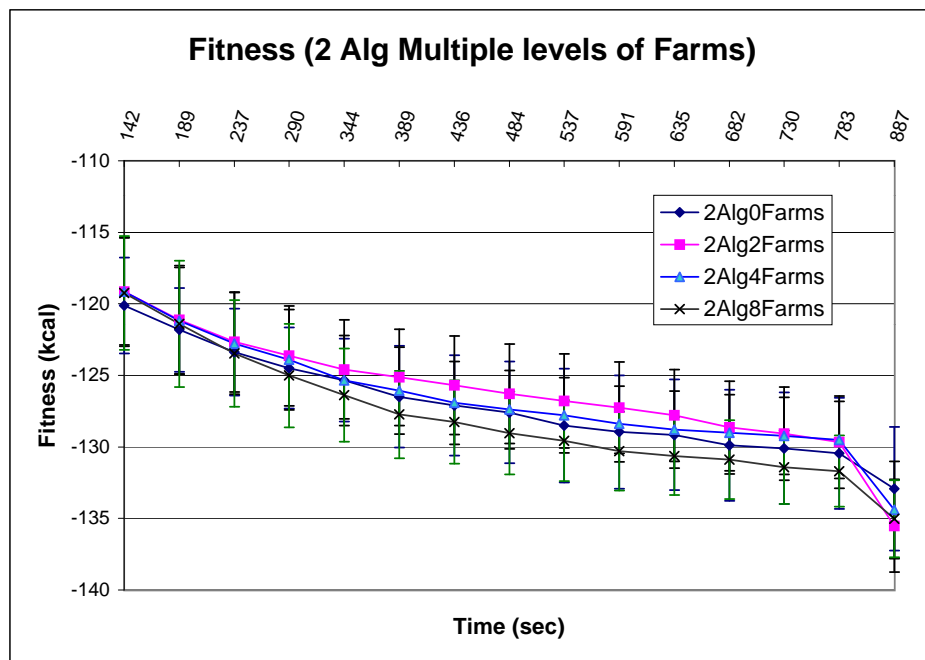
A	4	10	87.910881	-22.85427	-28.982138	-29.933531	-28.51818
A	4	6	38.873369	-23.216853	-29.087247	-29.93892	-28.560447
A	4	7	17.501835	-24.662159	-29.18398	-29.951425	-28.579357
A	4	8	15.511864	-25.100213	-29.272844	-29.959467	-28.590051
A	4	9	15.189762	-25.629695	-29.481002	-29.971166	-28.615457
A	4	10	14.961465	-25.945673	-29.632564	-29.981395	-28.620046
A	4	6	14.941421	-26.513462	-29.791052	-30.10208	-28.624681
A	4	7	14.910948	-26.729088	-29.925289	-30.240974	-28.632895
A	4	8	14.809754	-26.864212	-29.961172	-30.305014	-28.641967
A	4	9	14.711298	-26.918393	-30.025029	-30.321806	-28.672372
A	4	10	-29.17127	-27.909507	-30.671037	-31.006006	-28.926281
A	5	6	-30.121645	-9.337104	-27.250624	-29.104712	-28.250655
A	5	7	-30.272421	-9.464636	-27.301467	-29.124895	-28.253292
A	5	8	-30.423159	-9.488237	-27.330906	-29.139452	-28.25493
A	5	9	-30.515702	-9.509529	-27.649159	-29.165999	-28.257215
A	5	10	-30.57726	-9.534357	-27.784653	-29.205454	-28.258735
A	5	6	-30.659802	-9.580557	-27.962126	-29.239348	-28.259766
A	5	7	-30.701935	-9.635807	-28.008511	-29.257356	-28.263435
A	5	8	-30.753226	-9.654157	-28.033612	-29.27559	-28.264618
A	5	9	-30.788998	-9.670796	-28.063183	-29.28794	-28.266738
A	5	10	-30.811702	-9.759652	-28.112512	-29.308784	-28.271522
A	5	6	-30.826833	-9.878034	-28.135888	-29.326165	-28.275765
A	5	7	-30.8614	-11.683124	-28.144672	-29.335784	-28.279479
A	5	8	-30.87636	-14.772554	-28.155536	-29.346989	-28.284411
A	5	9	-30.915044	-17.182775	-28.170704	-29.36475	-28.289892
A	5	10	-31.715513	-24.299745	-29.088288	-30.755775	-28.490466
A	6	6	-24.959809	-25.520963	-22.664674	-27.772418	-28.951607
A	6	7	-25.099421	-25.567363	-23.090499	-28.514888	-29.014348
A	6	8	-25.210482	-25.643034	-23.577252	-28.83857	-29.06538
A	6	9	-25.510843	-25.724013	-23.612413	-29.225132	-29.095258
A	6	10	-25.639811	-25.739837	-23.657753	-29.534768	-29.130386
A	6	6	-25.805228	-25.746187	-23.66805	-29.660103	-29.162115
A	6	7	-25.919119	-25.898756	-23.687378	-29.734617	-29.178253
A	6	8	-25.994865	-26.008221	-23.698292	-29.792857	-29.369362
A	6	9	-26.117577	-26.172263	-23.7129	-29.932776	-29.985312
A	6	10	-26.338183	-26.254907	-23.723091	-29.990284	-30.284043
A	6	6	-26.436053	-26.306821	-23.730327	-30.008719	-30.40848
A	6	7	-26.511121	-26.40312	-23.738767	-30.041085	-30.474269
A	6	8	-26.764143	-26.4333	-23.752405	-30.107225	-30.562774
A	6	9	-26.844581	-26.478641	-23.758425	-30.13061	-30.646331
A	6	10	-30.811512	-26.976863	-23.852029	-30.556677	-31.367586
A	7	6	-27.859449	-25.740542	-27.957889	-28.873577	-27.540423
A	7	7	-27.946052	-25.942204	-27.960599	-28.961971	-27.56524
A	7	8	-27.974035	-26.118785	-27.961785	-29.088642	-27.583422
A	7	9	-28.092053	-26.397452	-27.964526	-29.26427	-27.59965
A	7	10	-28.363535	-26.517013	-27.966853	-29.457894	-27.607729
A	7	6	-28.818672	-26.648663	-27.968204	-29.815984	-27.619557
A	7	7	-29.1646	-26.767441	-27.968944	-29.968379	-27.630014
A	7	8	-29.401416	-27.015739	-27.969992	-30.150273	-27.652205
A	7	9	-29.608784	-27.146787	-27.970661	-30.253471	-27.661772
A	7	10	-29.673834	-27.233437	-27.97113	-30.278196	-27.670112

A	7	6	-29.691154	-27.284272	-27.971851	-30.315802	-27.683672
A	7	7	-29.722259	-27.328041	-27.972659	-30.351816	-27.693424
A	7	8	-29.786176	-27.345828	-27.973844	-30.458468	-27.716698
A	7	9	-29.811094	-27.376755	-27.974832	-30.554153	-27.746247
A	7	10	-30.656944	-29.947804	-27.985743	-31.834146	-28.463558
A	8	6	-3.187006	-23.067095	-31.353804	-28.752841	-27.70881
A	8	7	-13.105512	-23.294872	-31.387973	-28.847143	-27.794294
A	8	8	-20.374269	-23.757603	-31.408582	-28.99429	-27.844671
A	8	9	-22.457411	-24.652402	-31.452689	-29.198772	-27.937523
A	8	10	-25.393834	-24.945233	-31.467282	-29.897741	-27.976135
A	8	6	-26.018754	-25.151915	-31.486472	-30.007544	-28.0406
A	8	7	-26.278669	-25.380616	-31.490278	-30.066872	-28.091857
A	8	8	-26.68531	-25.566187	-31.494151	-30.118277	-28.117874
A	8	9	-26.816202	-25.742466	-31.510526	-30.160348	-28.168257
A	8	10	-27.040877	-25.850273	-31.520081	-30.241251	-28.232768
A	8	6	-27.117978	-25.922488	-31.535469	-30.394412	-28.730294
A	8	7	-27.198792	-25.974682	-31.540702	-30.473554	-28.80151
A	8	8	-27.292018	-26.015022	-31.545872	-30.554933	-28.872024
A	8	9	-27.367124	-26.082664	-31.559477	-30.582657	-28.935165
A	8	10	-27.831352	-26.598016	-33.191229	-31.141103	-30.59044
A	9	6	-28.529356	-27.789909	-22.333956	-27.720581	-28.537503
A	9	7	-28.540691	-28.191919	-23.811184	-27.770346	-28.570848
A	9	8	-28.5569	-29.779867	-24.527085	-27.819714	-28.743681
A	9	9	-28.565496	-30.731539	-24.993939	-27.83995	-29.607881
A	9	10	-28.577562	-31.096017	-25.149024	-27.860377	-29.698347
A	9	6	-28.586399	-31.352341	-25.195396	-27.878952	-29.83105
A	9	7	-28.600188	-31.452045	-25.253203	-27.927564	-29.937396
A	9	8	-28.605769	-31.56236	-25.337427	-27.940553	-29.961474
A	9	9	-28.618286	-31.737161	-25.383813	-27.957921	-30.021398
A	9	10	-28.6247	-31.859242	-25.428261	-27.976722	-30.063744
A	9	6	-28.635353	-31.897216	-25.451627	-27.99271	-30.104804
A	9	7	-28.679264	-31.985131	-25.636005	-28.00372	-30.203275
A	9	8	-28.698018	-32.089786	-25.689905	-28.030261	-30.244135
A	9	9	-28.718824	-32.390163	-25.722414	-28.040802	-30.274862
A	9	10	-29.046477	-34.113693	-27.451837	-29.905367	-30.876003
A	10	6	-27.146776	-27.174139	-26.373158	-28.718501	-26.371948
A	10	7	-27.397523	-27.197578	-26.416674	-28.862021	-26.377004
A	10	8	-27.471102	-27.213815	-26.456726	-28.950068	-26.398372
A	10	9	-27.610065	-27.239311	-26.483824	-29.032143	-26.893681
A	10	10	-27.745492	-27.2592	-26.505667	-29.094624	-27.675271
A	10	6	-28.015673	-27.274407	-26.517281	-29.139516	-28.694252
A	10	7	-28.084082	-27.282469	-26.537375	-29.175454	-29.660263
A	10	8	-28.130935	-27.295362	-26.549349	-29.205994	-29.996044
A	10	9	-28.273779	-27.312706	-26.565913	-29.225646	-30.232116
A	10	10	-28.386679	-27.325889	-26.579933	-29.288731	-30.518
A	10	6	-28.468874	-27.354299	-26.588499	-29.341649	-30.624207
A	10	7	-28.56598	-27.378948	-26.593336	-29.379428	-30.746158
A	10	8	-28.616878	-27.388348	-26.603891	-29.502677	-30.777375
A	10	9	-28.69106	-27.399102	-26.801087	-29.674032	-30.836572
A	10	10	-29.392082	-30.466633	-27.321161	-30.46088	-31.545856

Appendix M. Farming Model Experiment Graphs and Table

Table M.1 Computer Systems		
# of Computers	Processor	Operating System
8	1.7GHz P-IV	Linux V7.1
2	1.2 GHz P-III	Linux V7.1
5	1.0 GHz P-III	Linux V6.2
2	933MHz	Linux V6.2
4	450MHz	Linux V6.2





Appendix N. Atom and Amino Acid Identification

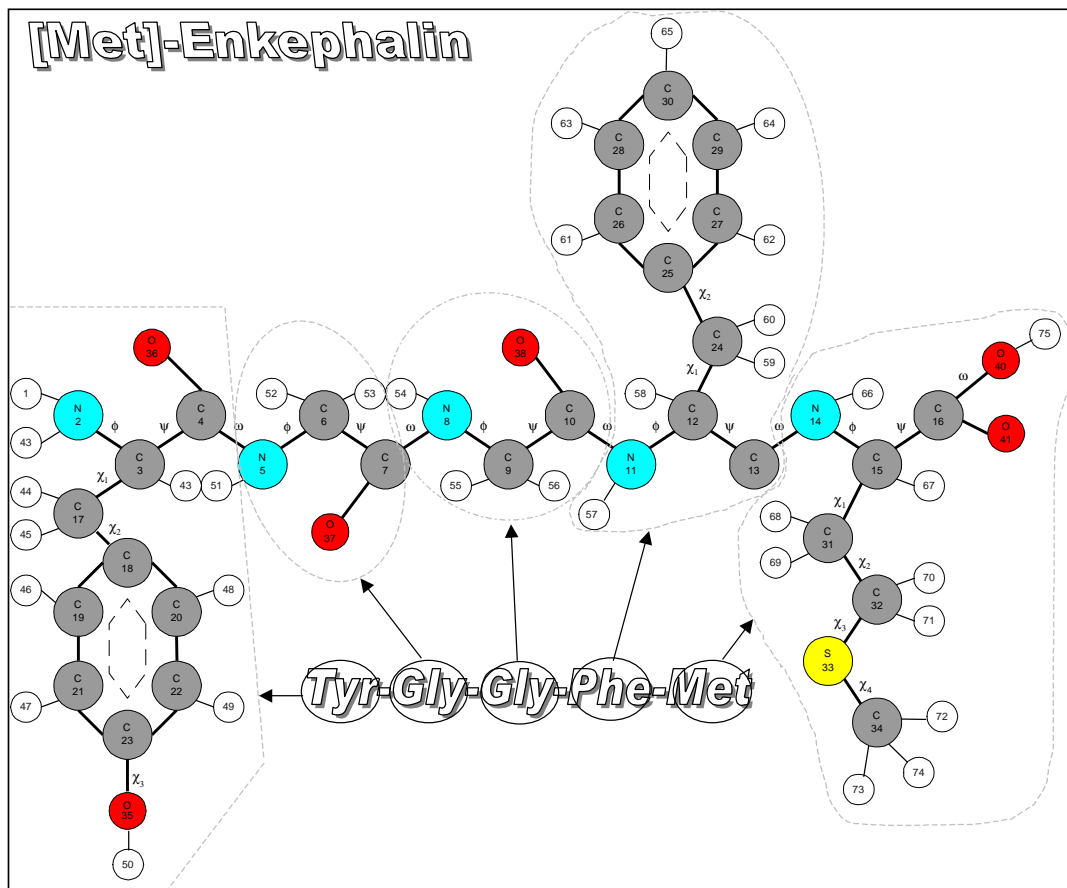


Figure N.1 MET's Amino Acid and Atom number identification figure.

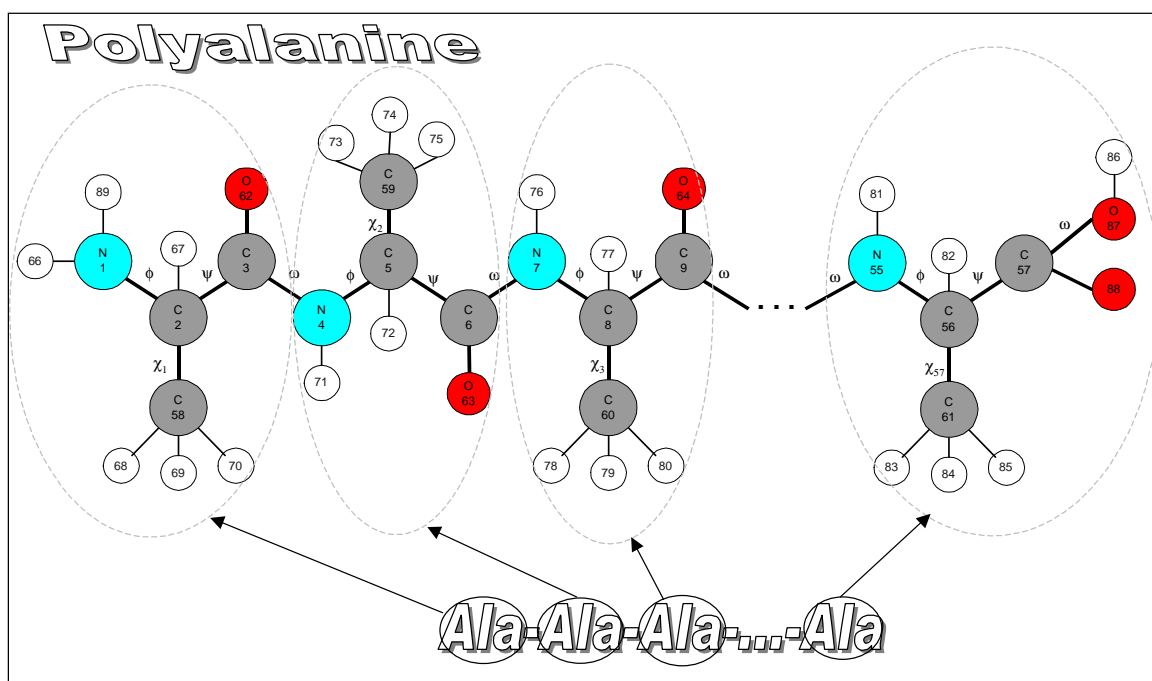


Figure N.2 PLOY's Amino Acid and Atom number identification figure.

Bibliography

- [1] Al-Attar, Akeel A. "A Hybrid GA-Heuristic Search Strategy," *AI EXPERT USA* (SEPTEMBER 1994). Miller Freeman.
- [2] Allinger, N. L. *Journal of American Chemistry Society*, 81:5727 (1959).
- [3] Bäck, T., et al. *Evolutionary Computation 1* (1st Edition), 1. Institute of Physics, 2000.
- [4] Bäck, Thomas. *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press, 1996.
- [5] Baltzer, Lars, et al. "De Novo Design of Proteins—What are the Rules?," *American Chemical Society*, 101:3153–3163 (2001).
- [6] Bell, Colin R. "Microbiology Mutations." plato.acadiau.ca/courses/biol/Microbiology/mutation.htm, November 2001.
- [7] Bindewald, Eckart, et al. "Implementing Genetic Algorithms with Sterical Constraints for Protein Structure Prediction," *5th International Conference Parallel Problem Solving from Nature – PPSN V*, 1(1498):959–967 (1998). Lehrstuhl für Informatik V, Universität Mannheim 68131 Mannheim, Germany.
- [8] Branden, Carl and John Tooze. "Introduction to Protein Structure," (1991).
- [9] Brinkman, Donald J. *Genetic algorithms and their application to the protein folding problem*. MS THESIS, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, December 1993.
- [10] Brinkman, Donald J., et al. "Parallel Genetic Algorithms and Their Application to the Protein Folding Problem," *Intel Supercomputer Users Group Annual Meeting* (October 1993).
- [11] Brooks, Bernard, et al. *CHARMm A Program for Macromolecular Energy, Minimization, and Dynamics Calculations*, 4, 187–217. John Wiley and Sons, INC, 1983.
- [12] Bryngelson, Joseph D., et al. *From Interatomic Interactions to Protein Structure*, 480, chapter Physics of Biological Systems : From Molecules to Species, 80–116. Springer-Verlag New York, 1997.
- [13] Cahoon, J.P., et al. "A multi-population genetic algorithm for solving the k-partition problem on hyper-cube," *Proceedings of the Fourth International Conference on Genetic Algorithms, San Mateo, CA*, pages 244–248 (July 1991). In Richard K. Belew and Lashon B. Booker, editors.

- [14] Committee on Physical, Mathematical and Engineering Sciences. *Grand Challenges 1993: High Performance Computing and Communications*. Office of Science and Technology Policy, 1982.
- [15] David Brown, MD, Washington Post Staff Writer. “Deciphering The Message of Life’s Assembly,” *The Washington Post* (1999).
- [16] Day, Richard O., et al. “Competitive Template Analysis of the Fast Messy Genetic Algorithm When Applied to the Protein Structure Prediction Problem,” *ICCN*, 4 (December 22 2001).
- [17] Day, Richard O., et al. “Solving the Protein Structure Prediction Problem Through a Multiobjective Genetic Algorithm,” *ICCN*, 4 (December 22 2001).
- [18] Day, Richard O., et al. “Genetic Algorithm Approach to Protein Structure Prediction with Secondary Structures,” *EUROGEN*, 6 (September 2000).
- [19] Day, Richard O., et al. “Analysis of Fine Granularity in Parallelization and Building Block Sizes of the Parallel Fast Messy GA used on the Protein Structure Prediction Problem,” *World Congress on Computational Intelligence*, 6 (December 2001). Special Biological area.
- [20] Deerman, Karl R. *Protein Structure Prediction Using Parallel Linkage Investigating Genetic Algorithms*. MS THESIS, Air Force Institute of Technology, Wright Patterson AFB, OH, March 1999. Sponsor: AFRL/Material Directorate.
- [21] Deerman, Karl R., et al. “Linkage Investigation Genetic Algorithms and Their Application of the Protein Structure,” *ACM Symposium on Applied computing (SAC01)* (March 11-14 2001). Las Vegas, Nevada.
- [22] Derrida, B. “Random Energy Model: Limit of a Family of Disordered Models.” *Phys. Rev. Lett.* 45. 1980.
- [23] Duntz, Shannon K., et al. *Petaflop Computing for Protein Folding*. Project Report, International Business Machines Corporation, 2001.
- [24] Ecker, J.G., et al. “An application of nonlinear optimization in molecular biology,” *European Journal Of Operational Research*, 138(2):452–458 (April 2002). Department of Mathematical Sciences, Rensselaer Polytechnic Institute, 110 8th Street, Troy, NY 12180-3590, USA.
- [25] et al Soukhanov. *WEBSTER’S II New Riverside University Dictionary*. Boston, MA: The Riverside Publishing Company, 1984.
- [26] Fagan, Phoebe, et al. *Protein Data Bank Annual Report 2001*. Research Collaboratory for Structural Bioinformatics, San Diego SuperComputer Center at the University of CA, San Diego: National Institute of CA, San Diego, June 2000-2001. Rutgers, The State University of New Jersey.

- [27] Fanford, David J. "X-ray crystography." Per conversation, 25 Feburary 2002.
- [28] Fonseca, Carlos M. and Peter J. Fleming. "Multiobjective Optimization." *Evolutionary Computation 2*, edited by Thomas Bäck, et al., 25–37, Institute of Physics Publishing, 2000.
- [29] Forman, Sean Lorell. *Torsian Angle Selection and Emergent Non-Local Secondary Structure in Protein Structure Prediction*. Thesis submitted in partial fulfillment of req. of PhD, College of The University of Iowa, December 2001. Thesis Advisor, Professor Alberto Maria Segre.
- [30] Fried, George H. and George J. Hademenos. *Schaum's Outline: Biology* (2nd Edition), 1. McGraw-Hill, 1999.
- [31] Fried, George H. and George J. Hademenos. *Schaum's Outline: Biology* (2nd Edition), 1. McGraw-Hill, 1999.
- [32] Gates, George H. *Predicting Protein Structure using Parallel Genetic Algorithms*. MS THESIS, Air Force Institute of Technology, Wright Patterson AFB, OH, December 1994. Sponsor: AFOSR WL/Material Directorate.
- [33] Gates, George H., et al. "Simple Genetic Algorithm parameter Selection for Protein Structure Prediction," *International Conference on Evolutionary Algorithms* (December 1995). Perth, Austria.
- [34] Gates, George H., et al. "Parallel Simple GAs vs Parallel Fast Messy GAs for Protein Floding Problem," *Intel Supercomputer Users Groupt Annual Meeting* (June 1995).
- [35] Gaulk, Robert L. *The Application of Hybridized Genetic Algorithms to the Protein Folding Problem*. MS Thesis, Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio, December 1995.
- [36] Gibson, K. D. and H. A Scheraga *Biopolymers*, 4:709 (1966).
- [37] Go, Nobuhiro and Harold A. Scheraga. "Calculation of the Conformation of Pentapeptide cyclo-(Glycylglycylglycylprolylprolyl). I. AComplete Energy Map," *Macromolecules*, 3:188–194 (December 1969). Department of Chemistry, Cornell University, Ithaca, New York.
- [38] Go, Nobuhiro and Harold A. Scheraga. "Ring Closure and Local Conformational Deformation of Chain Molecules," *Macromolecules*, 3:178–187 (December 1969). Department of Chemistry, Cornell University, Ithaca, New York.
- [39] Go, Nobuhiro and Harold A. Scheraga. "Ring Closure and Local Conformational Deformations of Chain Molecules," 178–187 (December 4 1969).
- [40] Goldberg, David E. *Genetic Algorithms in Search, Optimization and Machine Learning*. Reading MA: Addison-Wesley Publishing Company, 1989.

- [41] Goldberg, David E. "Messy Genetic Algorithms Revisited: Studies in Mixed Size and Scale," *Complex Systems*, 415–444 (1990).
- [42] Goldberg, David E., et al. "Rapid, Accurate Optimization of Difficult Problems Using Fast Messy Genetic Algorithms," 56–64 (July 1993).
- [43] Goldberg, David E., et al. "Compressed Introns in a Linkage Learning Genetic Algorithm," (December 1997).
- [44] Hart, William E. and Sorin Istrail. "Fast protein folding in the hydrophobic-hydrophilic model within three-eighths of optimal (Extended Abstract)." *ACM Symposium on Theory of Computing*. 157–168. 1995.
- [45] Heermann, D. W. *Computer Simulation Methods in Theoretical Physics*. Springer-Verlag, 1990.
- [46] Holland, John J. *Adaptation in Natural and Artificial Systems*. Ann Arbor MI: The University of Michigan: The University of Michigan Press, 1975.
- [47] Jain, Raj. *The Art of Computer Systems Performance Analysis*. Wiley, 1991.
- [48] James, Thomas L. *Fundamentals of NMR*. Technical Report, San Francisco, CA 94143: Department of Pharmaceutical Chemistry, University of CA, 1998. Copywrite by author.
- [49] Janssen, Deborah. "Putting the Proteome Into Perspective," *Genomics and Proteomics*, 28–30 (January/February 2002). www.genpromag.com.
- [50] Kaiser, Charles E. *Refined Genetic Algorithms for PolyPeptide Structure Prediction*. MS THESIS, Air Force Institute of Technology, Wright Patterson AFB, OH, December 1996. Sponsor: AFRL/Material Directorate.
- [51] Kaiser, Charles E., et al. "Real Valued Hybrid Genetic Algorithms for Polypeptide Structure Prediction," *28th American Chemical Society Central Regional Meeting* (June 1996). Dayton, Ohio.
- [52] Kaiser, Charles E., et al. "Exploiting Domain Knowledge in Genetic Algorithms for Polypeptide Structure Prediction (Abstract)," *212th American Chemical Society National Meeting* (August 1996). Orlando, Florida.
- [53] Kaiser, Charles E., et al. "Real-valued Genetic Algorithm Case Studies in Protein Structure Prediction," *SIAM Conference on Parallel Processing for Scientific Computing* (March 1997). Minneapolis, Minnesota.
- [54] Kaiser, Charles E., et al. "Polypeptide Structure Prediction: Real-Valued versus Binary Hybrid Genetic Algorithms," *ACM Symposium on Applied Computing* (February 28-March 2 1997). Atlanta, Georgia.
- [55] Kaiser, Charles E., et al. "Exogenous Parameter Selection in Real-Valued Genetic Algorithms," *International Conference on Evolutionary Computation (ICE97)* (April 1997). Indianapolis, Indiana.

- [56] Khimasia, Mehul M. and Peter V. Coveney. "Protein structure prediction as a hard optimization problem: the genetic algorithm approach," (June 1997). Theory of Condensed Matter, Cavendish Laboratory.
- [57] Khimasia, Mehul M. L. "NP Complete Problems." <http://www.tcm.phy.cam.ac.uk/~mmlk2/report13/node31.html>, 1996.
- [58] Kollman, Peter A. "Department of Pharmaceutical Chemistry, University of California San Francisco." Energy Function.
- [59] Kono, Hidetoshi and Jeffery G. Saven. "Statistical Theory for Protein Combinatorial Libraries. Packing Interactions, Backbone Flexibility , and the Sequence Variability of a Main-chain Structure," *Journal of Molecular Biology*, 306:607–628 (2001). doi:10.1006/jmbi.2001.4422.
- [60] Kotelyanskii, Michael. "Off-lattice Monte-Carlo Simulations." Penn State Polymer Physics Group, December 1997.
- [61] Krasnogor, Natalio, et al. "Enhanced Evolutionary Search of Folding Using Parsed Proteins," *Operational Research Symposium* (1997). Bs. As. Argentina.
- [62] Kronsjo, Lydia and Dean Shumsheruddin (editors). *Advances in Parallel Algorithms*. Halsten Press, New York, 1992.
- [63] Kumar, V. *Introduction to Parallel Computing* (1st Edition). Redwood City: Benjamin/Cummings Publishing Company, Inc, 1994.
- [64] Laboratories, Lawrence Livermore. *Blue Gene Project Update*. Los Alamos Sandia and Lawrence Livermore Laboratories: Project Report, International Business Machines Corporation, January 2002.
- [65] Lamont, Gary B. and Laurence D. Merkle. "Introduction to Bioinformatics for Computer Scientists." Chapter in W. Corne's book, August 2002.
- [66] Lengauer, Thomas. "Algorithmic Research Problems in Molecular Bioinformatics," *Arbeitspapiere der GMD 748* (May 1993).
- [67] McColl, W. F. "Scalable Computing." *Computer Science Today: Recent Trends and Developments 1000*. LNCS, edited by J. van Leeuwen. 46–61. Springer-Verlag, 1995.
- [68] Merkle, Laurence D. *Generalization and Parallelization of Messy Genetic Algorithms and Communication in Parallel Genetic Algorithms*. Thesis, Air Force Institute of Technology, December 1992. Sponsor: WL/Material Directorate.
- [69] Merkle, Laurence D., et al. "Scalability of an MPI-Based Fast Messy Genetic Algorithm," *ACM Symposium on Applied Computing (SAC98)* (February 1998). San Antonio, Texas.

- [70] Merkle, Laurence D., et al. "Parallel Messy Algorithms for the Protein Folding Problem," *Proceedings of Intel Supercomputing Users Group Annual Meeting* (June 1994). pp. 189-195.
- [71] Merkle, Laurence D., et al. "Hybrid Genetic Algorithms for Polypeptide Energy Minization," *ACM Symposium on Applied Computing* (February 1996). Philadelphia, Pennsylvania.
- [72] Merkle, Laurence D., et al. "Hybrid Genetic Algorithms for Minimization of a Polypeptide Specific Energy Model," *IEEE Conference on Evolutionary Computation* (May 1996). Japan.
- [73] Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs* (2nd Edition). Springer-Verlag, 1994.
- [74] Michaud, Steven R. *Protein Structure Prediction using Refined Parallel Fast Messy Genetic Algorithms*. MS THESIS, Air Force Institute of Technology, March 2001. Sponsor: AFRL/Material Directorate.
- [75] Michaud, Steven R., et al. "Scaling a Genetic Algorithm to Medium-Sized Peptides by Detecting Secondary Structures with an Analysis of Building Blocks." *Proceedings of the First International Conference on Computational Nanoscience*, edited by Matthew Laudon and Bart Romanowicz. 29–32. March 2001.
- [76] Michaud, Steven R., et al. "Load Balancing the Parallel Fast Messy Genetic Algorithm for Increased Computational Efficiency in Attempting to Solve the Protein Structure Prediction Problem with a Heterogeneous Cluster of PCs," *Tenth SIAM Conference on Parallel Processing for Scientific Computing (PP'01)* (March 12-14 2001). Portsmouth, Virginia.
- [77] Michaud, Steven R., et al. "Scaling a Genetic Algorithm to Medium-Sized Peptides by Detecting Secondary Structures with an Analysis of Building Blocks," *First International Conference on Computational Nanoscience (ICCN'01)* (March 19-21 2001). Hilton Head Island, South Carolina.
- [78] Moscato, Pablo. *New Ideas in Optimization*, chapter 14, 219–234. McGraw-Hill, 1999.
- [79] Neumaier, A., et al. "New Techniques for the Construction of Residue Potentials for Protein Folding." 212–226. 1998.
- [80] Neumaier, Arnold. "Molecular Modeling of Proteins and Mathematical Prediction of Protein Structure," *SIAM Review*, 39(3):407–460 (1997).
- [81] Nielson, Gregory M. *Scientific Visualization*. Los Alamitos, CA: Matt Loeb, 1997.
- [82] of Health (United States), National Institutes, "NHGRI Workshop on DNA Sequence Validation." Online National Human Genome Research Institute,

April 1996. National Advisory Council for Human Genome Research report addresses validation issues.

- [83] P. Grassberger, W Nadler, G T Barkema. *Monte Carlo Approach to Biopolymers and Protein Folding*. November 1998. John von Neumann-Institut für Computing.
- [84] Piccolboni, Antonio and G. Mauri. “Application of Evolutionary Algorithms to Protein Folding Prediction.” *Artificial Evolution*. 123–136. 1997.
- [85] Piccolboni, Crescenzi Goldman Papadimitriou and Yannakakis. *On the Complexity of Protein Folding*.
- [86] Quackenbush, John, et al. “Reconstruction and Annotation of Transcribed Sequences: The TIGR Gene Indices,” *DOE Human Genome Program, Contractor-Grantee Workshop VIII (Santa Fe, NM)* (February 27-March 2 2000). The Institute for Genomic Research, Rockville, MD 20850.
- [87] Ripoll, Daniel R. “Electrostatically Driven Monte Carlo (EDMC) Protein Folding.” Baker Lab. of Chemistry.
- [88] Rupp, Bernhard. “Protein Crystallization.” Lawrence Livermore National Laboratory, University of California, LLNL-BBRP, Livermore, CA 94551.
- [89] Saven, Jeffery G. “Designing Protein Energy Landscapes,” *American Chemical Society*, 101:3113=3130 (2001).
- [90] Schulze-Kremer, Steffen. “Genetic Algorithms and Protein Folding.” Westfälische Strasse 56, D-10711 Berlin, FRG, June 1996.
- [91] Shafner, J. P. *The Science and Design of Engineering Material* (2nd Edition). New York: McGraw-Hill, 1998.
- [92] Shanmugan, K. Sam and A. M. Breipohl. *Random Signals Detection, Estimation and Data Analysis*, chapter 3, 126. Wiley, 1988.
- [93] Spiegel, Murray R. and Larry J. Stephens. *Theory and Problems of Statistics*, 1. 3rd. McGraw-Hill, 1999.
- [94] Stryer, Lubert. *Biochemistry* (4th Edition), 1. 2nd Printing. New York: W.H. Freeman and Company, 1995.
- [95] Syswerda, G. *Handbook of Genetic Algorithms*, chapter Schedule optimization using genetic algorithms, 332–49. New York: Van Nostrand Reinhold, 1991. ed. L Davis.
- [96] University, Harvard. “CHARMm energy functions.” http://www.ch.embnet.org/MD_tutorial/pages/MD.Part2.html, 2001.
- [97] Van Veldhuizen, David A. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD dissertation, Department of

Electrical and Computer Engineering. Graduate School of Engineering. Air Force Institute of Technology, Wright-Patterson AFB, Ohio, May 1999.

- [98] Van Veldhuizen, David A. and Gary B. Lamont. *Multiobjective Evolutionary Algorithm Research: A History and Analysis*. Technical Report TR-98-03, Wright-Patterson AFB, Ohio: Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, 1998.
- [99] Venkatraman, Janani, et al. "Design fo Folded Peptides," *American Chemical Society*, 101:3131=3152 (2001).
- [100] Wendl, Chris. *Domain Decomposition using Parallel Genetic Algorithms*. Paper, James Clerk Maxwell Building, The Kings Buildings, The University of Edinburgh, Mayfield Road, Edinburgh EH9 3JZ, Scotland: Edinburgh Parallel Computing Centre, 1996.
- [101] Whitley, Darrel. "A genetic algorithm tutorial," *Statistics and Computing*, 4:65–85 (1994).
- [102] Wiberg, K. B. *Journal of American Chemistry Society*, 87:1070 (1965).
- [103] Zhang, Shuguang, et al. "Spontaneous Assembly of a Self-Complementary Oligopeptide to Form a Stable Macroscopic Membrane." Massachusetts Institute of Technology, Cambridge, MA 02139, National Academy of Sciences 1992.
- [104] Zydallis, Jesse B., et al. "A Statistical Comparison of Multiobjective Evolutionary Algorithms Including the MOMGA–II." *First International Conference on Evolutionary Multi-Criterion Optimization* edited by Eckart Zitzler, et al., 226–240, Springer-Verlag. Lecture Notes in Computer Science No. 1993, 2001.

Vita

Captain Richard Orison Day enlisted in the United States Air Force in December 1988. He was assigned to the 611th Aerial Port Squadron, Osan Air Base in Korea where he began taking classes through the University of Maryland's Asian Division. In May of 1993 he had a permanent change of duty stations and was assigned to the 1st Communications Squadron, Langley Air Force Base, Virginia where he continued to take classes at Christopher Newport University, Newport News, VA. In 1995, he applied and was accepted into the Airman Education and Commissioning Program (AECPP). Directly following his acceptance into this competitive program he attended Clemson University earning a Bachelors of Science in Computer Engineering in August of 1997. He was commissioned through Officer's Training School in December of that same year. Following commissioning he completed a short Acquisition Officer's course at Lackland Air Force Base, Texas and was assigned to the National Air Intelligence Center at Wright Patterson Air Force Base (WPAFB), Ohio where he was the 1st officer and computer engineer assigned to the the Open Skies Treaty Media Processing Facility (OSMPF). Then, Lt Day, planned, organized, and executed the placement of computer systems and media processing computer support systems and software for the entire operation at the OSMPF. He then left Air Intelligence Command to attend AFIT where he is expected to graduate in March 2002. Afterward, Captain Day will be continuing his education in pursuit of a Doctorial of Philosophy degree at AFIT in signal processing.

REPORT DOCUMENTATION PAGE				Form Approved OMB No. 0704-0188	
<p>The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing the burden, to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to any penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.</p> <p>PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.</p>					
1. REPORT DATE (DD-MM-YYYY) 07-03-2002		2. REPORT TYPE Maters Thesis		3. DATES COVERED (From - To) August 2000 - March 2002	
4. TITLE AND SUBTITLE A MULTIOBJECTIVE APPROACH APPLIED TO THE PROTEIN STRUCTURE PREDICTION PROBLEM				5a. CONTRACT NUMBER	
				5b. GRANT NUMBER	
				5c. PROGRAM ELEMENT NUMBER	
				5d. PROJECT NUMBER AFIT/GE/ENG/02M-05	
6. AUTHOR(S) Day, Richard O.				5e. TASK NUMBER	
				5f. WORK UNIT NUMBER	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Dr Gary B. Lamont AFIT/ENG Gary.Lamont@afit.edu				8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Dr. Ruth Pachter Materials and Manufacturing Directorate Air Force Research Laboratory WPAFB, OH 45433 Ruth.Pachter@wpafb.af.mil 785-3808 x3177				10. SPONSOR/MONITOR'S ACRONYM(S)	
				11. SPONSOR/MONITOR'S REPORT NUMBER(S)	
12. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified, Unlimited					
13. SUPPLEMENTARY NOTES					
14. ABSTRACT <p>Interest in discovering a methodology for solving the Protein Structure Prediction problem extends into many fields of study including biochemistry, medicine, biology, and numerous engineering and science disciplines. Experimental approaches, such as, x-ray crystallographic studies or solution Nuclear Magnetic Resonance Spectroscopy, to mathematical modeling, such as minimum energy models are used to solve this problem. Recently, Evolutionary Algorithm studies at the Air Force Institute of Technology include the following: Simple Genetic Algorithm (GA), messy GA, fast messy GA, and Linkage Learning GA, as approaches for potential protein energy minimization. Prepackaged software like GENOCOP, GENESIS, and mGA are in use to facilitate experimentation of these techniques. In addition to this software, a parallelized version of the fmGA, the so-called parallel fast messy GA, is found to be good at finding semi-optimal answers in reasonable wall clock time. The aim of this work is to apply a Multiobjective approach to solving this problem using a modified fast messy GA. By dividing the CHARMM energy model into separate objectives, it should be possible to find structural configurations of a protein that yield lower energy values and ultimately more correct conformations.</p>					
15. SUBJECT TERMS Protein Structure Prediction problem, fast messy Genetic Algorithm, High Performance Computing					
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT UU	18. NUMBER OF PAGES 228	19a. NAME OF RESPONSIBLE PERSON Dr. Gary B. Lamont
a. REPORT U	b. ABSTRACT U	c. THIS PAGE U			19b. TELEPHONE NUMBER (Include area code) (937) 255-3636 x4718