AFIT/GCS/ENG/96D-13

REFINED GENETIC ALGORITHMS

FOR

POLYPEPTIDE STRUCTURE PREDICTION

THESIS

Charles Edward Kaiser, Jr.
First Lieutenant, USAF

AFIT/GCS/ENG/96D-13

AFIT/GCS/ENG/96D-13

REFINED GENETIC ALGORITHMS FOR POLYPEPTIDE STRUCTURE PREDICTION

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Computer Systems

Charles Edward Kaiser, Jr., BSIS

First Lieutenant, USAF

December, 1996

AFIT/GCS/ENG/96D-13

REFINED GENETIC ALGORITHMS FOR POLYPEPTIDE STRUCTURE PREDICTION

Charles Edward Kaiser, Jr., BSIS

First Lieutenant, USAF

Approved:

Gary B. Lamont, Advisor	
Eugene Santos, Committee Member	
Gregg H. Gunsch, Committee Member	
Sheila B. Banks, Committee Member	

Acknowledgements

I have benefited from the support of many people during this thesis effort. My sincerest thanks go to my thesis advisor, Dr Gary Lamont, who never balked when I pleaded for more time. His educational perspective motivated me to learn more than I thought possible. I'm indebted to my sponsor, Dr Ruth Pachter, for her support. By her giving me opportunities to present portions of this work before its completion, I've interacted personally with some of the great researchers in this field. I'd also like to acknowledge the other members of my committee, Dr Eugene Santos, Lt Col Gregg H. Gunsch, and Maj Sheila B. Banks.

Several students, former students and support staff members at AFIT have been more than helpful during the course of this investigation. Larry Merkle patiently endured my endless questions about genetic algorithms and the protein folding problem. On more than one occasion he realigned my perspective when I was unsatisfied with my work. In most discussions George Gates usually held the position opposite of mine. This force me to sharpen my saw. Jim Wager, our summer intern, helped by conducting many experiments for me. I'd also like to says thanks to Russ Milliron, who as system administrator, fixed problems almost as fast as I could create them. I'd also like to commend Jennifer Wedekind for outstanding administrative support and guidance.

None of my work would have been possible without the love and support of my family. Thank you, Charlie, PJ, and Michael for keeping your mommy company while I have been so busy. Finally, and most importantly, my deepest love and appreciation go to my wife, Michelle. I know her sacrifice has been much greater than mine. Thank you for your patience, devotion, and understanding.

Charles Edward Kaiser, Jr.

Table of Contents

List of Figures

List of Tables

List of Symbols

List of Abbreviations

Abstract

Accurate and reliable prediction of macromolecular structures, Polypeptide Structure Prediction (PSP), has eluded researchers for nearly 40 years. Prediction via energy minimization assumes the native conformation of the protein has the globally minimal energy potential. However an exhaustive search is impossible since for molecules of normal size, the size of the search space exceeds the accepted size of the universe (10^{80}, the number of stable elementary particles). Domain knowledge sources, such as the Brookhaven Protein Data Base and the Dictionary of Protein Secondary Structures in Germany, can be mined for constraints that limit the search space and possibly result in an efficient stochastic algorithm for PSP.

Genetic algorithms (GAs) are stochastic, population based, search algorithms of polynomial (P) time complexity that can produce semi-optimal solutions for problems of nondeterministic polynomial (NP) time complexity such as PSP. This study is an engineering investigating into performance (effectiveness & efficiency) gains from parallel and real-valued GAs with respect to PSP.

For PSP, the simple GA has previously been enhanced with local search techniques improving effectiveness. At the same time, run time increases by two orders of magnitude. A "farming model" parallel hybrid GA (PHGA) which preserves the effectiveness of the serial algorithm but with substantial speed up is designed and implemented. Portability across distributed and massively parallel platforms is accomplished with the Message Passing Interface (MPI) communications standard.

A real-valued GA system, the REal-valued Genetic Algorithm, Limited by constraints (REGAL), which exploits domain knowledge, is also designed and implemented. It integrates Michalewicz's real-valued GA for numerical optimization, AFIT/WL's own CHARMM energy model implementation and molecule model data structures, and constraint sets derived from the biochemistry domain. Results with derived sets of constraints are presented. Experiments with the pentapeptide Met-enkephalin have identified conformers with lower energies (CHARMM) than the accepted optimal conformer (Scheraga, et al), -31.98 vs -28.96 kcals/mol. Analysis of exogenous parameters yields additional insight into performance.

A parallel version, Para-REGAL, an "island model" modified to allow different active constraints in the distributed subpopulations is also designed and implemented. Results examining novel concepts of Probability of Migration and Probability of Complete Migration are presented.

REFINED GENETIC ALGORITHMS
FOR
POLYPEPTIDE STRUCTURE PREDICTION

I. Introduction

Most substantial problems cannot be mapped directly into a manageable, or tractable, computational model. Workable solutions to these problems necessarily involve some degree of abstraction. In Newtonian physics for instance, one might assume a frictionless pulley. Here, the problem is divided into a tractable portion with significant contributions, and an intractable portion that can be "safely" ignored. What constitutes an answer that is "close enough" and "safe" is problem specific. Tractable solution implementations involving digital computers are by necessity discrete, whereas many problem domains are continuous. Therefore, when a computer is used to solve a problem, some level of granularity must be accepted.

Even when there exists an acceptable level of granularity, it is not always possible to search for an optimal solution. The size of the solution space is roughly the average number of values the variables can assume raised by the number of degrees of freedom. Even with a linear growth in number of variables, the growth in solution space is exponential. The traditional search method of evaluating every possible solution cannot be done in a reasonable period of time, even with the fastest computers available today or in the future.

Thus, many classes of problems must be engineered to strike a balance between fidelity and solvability. This thesis investigation examines this issue with respect to one such problem, Polypeptide Structure Prediction.

1.1 Protein Folding Problem / Polypeptide Structure Prediction

The Protein Folding Problem (PFP) predicts the path taken by a protein (or polypeptide) transitioning from an unfolded (or denatured) state to its folded (or native) state (or conformation). A general solution to the protein folding problem has eluded researchers for more than 30 years (70). Polypeptides Structure Prediction (PSP) is a related problem. The essence of PSP is to *predict* the compact, three dimensional shape of a protein as it would exist in nature, without regard to path taken. This compact shape is called the *native conformation.*[1]

$$PSP \subseteq PFP \tag{1}$$

[1] The *native conformation* determines the protein's biological functions.

Equation 1 demonstrates the relationship between the two problem domains.

1.1.1 Background. A protein is a linear polymer molecule, a chain of tens to thousands of monomer units strung together like beads in a necklace. The monomers are the 20 naturally occurring amino acids (8). Each amino acid consists of a back plane formed by a single nitrogen, alpha-carbon (C_α), carbon, oxygen, and hydrogen atoms, and a distinguishing side chain (27). The backbone of a protein is a sequence of these back planes linked via peptide bonds. Because they exhibit this bond, proteins are a subset of molecules called polypeptides. In this document the term *polypeptide*, or just *peptide*, and *protein* is used interchangeably. In fact, one definition of a protein is a polypeptide with 50 or more amino acids. When an amino acid molecule joins with others to form a polypeptide, it splits out a water molecule (H_2O). The amino acid without the oxygen and hydrogen atoms is called a *residue*. In this document, the term "residue" refers to an amino acid in a sequence forming a particular polypeptide.

The *primary structure* is the sequence of amino acids making up a protein. The primary structure is easily determined using laboratory methods (27). In addition, the *secondary structure* is the shape of local sections of the primary structure. Common secondary structures include alpha-helices, beta-sheets, and random coils. These, too, are easily determined by sequence analysis and homology techniques. The large-scale architecture of a protein (how the helices, sheets, and other secondary structures fit together) is the *tertiary structure*. Determining this structure experimentally is very difficult and time consuming. First a crystalline structure must be grown. The crystal is then examined using x-ray crystallography or multidimensional nuclear magnetic resonance. Years are required to identify the conformation of a single protein (35). For comparison, the count of identified protein sequences is in the tens of thousands while only a few thousand conformations had been identified. Of these, only about 400 conformations had been determined to the level of atomic resolution (8). In addition, because the crystalline structure is itself an abstraction of the molecule from it native environment, it may not be the naturally occurring structure.

1.1.2 Importance. The PFP and PSP are fundamental problems in biophysical science (8), and are considered *Grand Challenges*. Grand Challenges are fundamental problems in science and engineering with broad economic and scientific impact whose solutions can be advanced by applying high performance computing techniques and resources (12). Understanding the PFP and PSP are of great importance for biomedicine: in designing novel proteins, in decoding the information obtained from the Human Genome Project (91), in designing new drugs, and in trying to understand the thousands of protein sequences being discovered everyday in biotechnology labs.

Efficient PSP techniques are also of great interest to the Materials Directorate of the USAF Wright Laboratory (WL). These techniques expedite their efforts to develop new materials. They intend to develop

materials with non-linear optical properties for the USAF. In particular, they plan to develop chromophore-substituted polymer chains with controlled optical properties, so-called smart filters or optical switches (93).

1.1.3 Methods for Polypeptide Structures Prediction. There are currently two basic techniques for predicting the tertiary structure of proteins: *energy minimization* and *molecular dynamics*. Using energy minimization, *ab initio* methods calculate the energy exactly, *semi-empirical* methods neglect some of the non-dominating terms, and *force-field* methods only account for pairwise interactions between atoms. Calculating a single energy value for these methods takes $\mathcal{O}(n^5)$, $\mathcal{O}(n^4)$, and $\mathcal{O}(n^2)$ time respectively (71). Also, it is assumed the native conformation is the least energy conformer corresponding to the global minimum of the energy function (or model). With this assumption, the objective becomes a search for the conformation with the least potential energy. This can be visualized for a rather trivial case of a n-butane molecule, Figure 1, by following the energy surface as the molecule is rotated about the bond between the central carbon atoms.

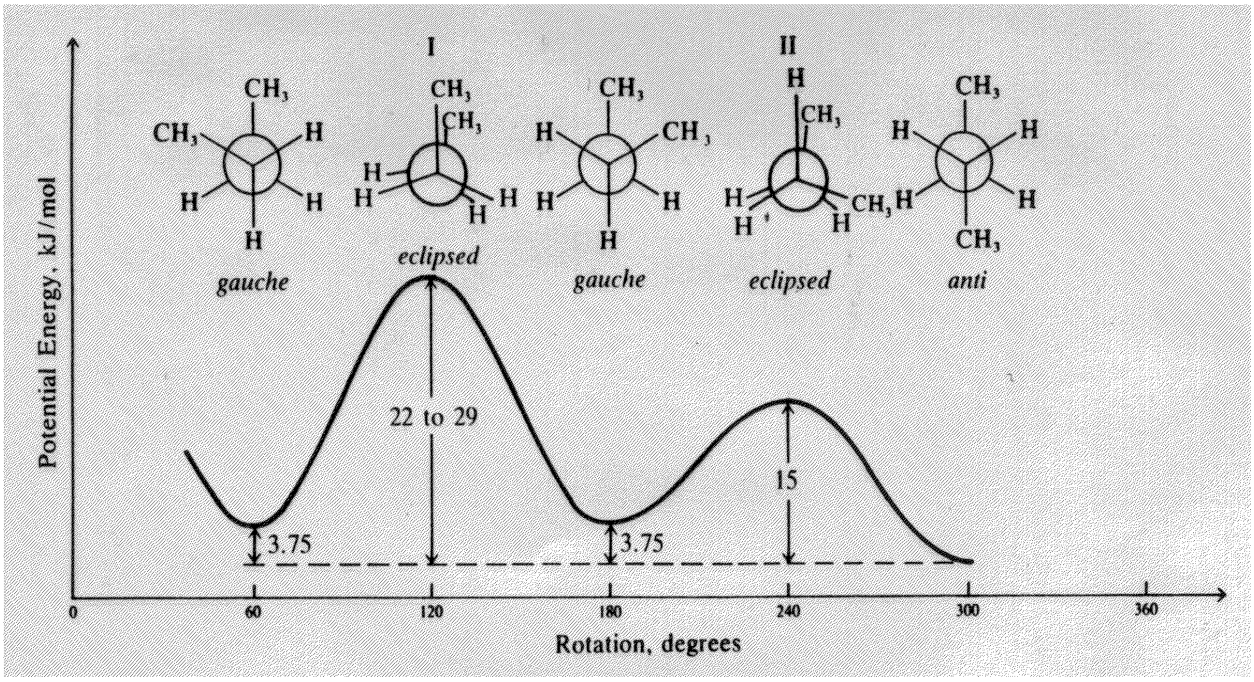


Figure 1. Potential energy trace for various conformers of n-butane
Adapted from Theory and Problems of Organic Chemistry by Meislich and others (76).

Molecular dynamics attempts to simulate the folding process. However, the time steps required for this simulation are on the order of one femtosecond (10^{-15} sec). Today's computers only allow us to simulate a few hundred picoseconds (10^{-12} sec), while the actual folding processes take at least microseconds to complete (71:5–7).

1.1.4 Growth of Complexity. There is an exponential growth in the solution space as the size of the molecule increases. First, there is a linear growth in the degrees of freedom (DOF) per atom added to the molecule. Each atom is treated as a symmetrical point mass, therefore, the three rotational DOF are ignored. This leaves the three translational DOF in the *x, y* and *z* planes. Also, the six DOF for the molecule as a whole are ignored since the focus is the molecule's internal interactions. Thus, there are $|3n-6|$ DOF where n is the number of atoms.[2] This leads to the size of the solution space: [number of values the variables can assume]$^{|3n-6|}$. Rarely is a search space of this magnitude conquered by classical methods, thus the interest in evolutionary and in particular genetic algorithms. This has been but a brief introduction to the biochemistry problems of Protein Folding and Structure Prediction. Additional information is provided in the Current Issues Chapter (Section 2.3), and Appendix A.

1.2 Genetic Algorithms

The Air Force Institute of Technology (AFIT) has a long history of research into computational science and engineering with particular interest in search algorithms and parallel and distributed computation. A Genetic Algorithm (GA) is characterized by *implicit parallelism*[3]. The study of GAs was a natural extension of the interest in both search and parallel algorithms. GA research has been carried on at AFIT for almost a decade(67). Much of that research has been sponsored by Wright Laboratory. In 1996, this research group adopted the name AFIT/WL Genetic Computation Techniques Research Group (AGCT).

GAs are stochastic semi-optimal search/optimization algorithms based on models of natural evolution (55) (41:1–2). GAs were developed in an attempt to create robust, semi-optimal search and optimization algorithms that would be applicable to a wide variety of problems (55, 67). They strike a balance between exploration and exploitation of the search space (84). GAs are one school of what has come to be called *Evolutionary Computing* (EC), which also includes Evolution Strategies (ES), Evolutionary Programming(EP), Genetic Programming (GP), and Classifier Systems (CS). A brief introduction to GAs follows. Additional information about EC and its variations, especially GAs, is available in the Current Issues Chapter (Section 2.4), and Appendix B. Other stochastic algorithms seen in the biochemistry community include Monte Carlo algorithms (74, 89) and Simulated Annealing (10, 15, 86, 89).

Genetic Algorithms work on populations of solutions. A candidate solution is called a *chromosome*[4] encoded as a string (of symbols). A Simple GA (SGA) performs three basic operations on the chromosomes:

[2] Regardless of whether a *Cartesian* or *Internal* coordinate system is used. However, the internal coordinate system has fewer *independent variables*

[3] See Appendix B.3.3

[4] Because GAs are loosely based on natural evolution, many of the terms associated with natural evolution are used interchangeably with the terms created specifically for genetic algorithms (67).

selection, crossover,[5] and mutation. The algorithm steps through these three operations repeatedly until some stopping criteria is met. The execution of a complete iteration $< selection,\ crossover,\ mutation >$ is called a *generation*. A chromosome is composed of *genes*, the algorithm space encoding of the problem space variables. *Alleles* are the values a gene can assume. Following the biological model, GAs have historically used a binary encoding. However, other encodings, such as real-valued or higher cardinality alphabets, are possible. Practitioners have observed better results for numerical optimization problem solutions with real-valued rather than binary encodings. Historically, at least at AFIT, GAs have been applied to PSP using a combinatoric paradigm that suggest a binary representation. If PSP is viewed instead as a numeric optimization problem, would a real-valued GA be more effective? The results of this research say **YES**.

1.3 Parallel and Distributed Computing

As we approach the physical limits of single processor computers, our attention has turned to parallel computer architectures for increased performance. Many architectures exist which exploit different parallelization schemes. The two primary architectures are *single instruction stream, multiple data stream* (SIMD) and *multiple instruction stream, multiple data stream* (MIMD) (66:16–17). Another growing area of parallel computing involves distributed computing on a network of workstations (NOW). Each of these architectures has benefits and limitations with respect to particular applications, communication and dependencies between tasks, and data and task distributions.

A major stumbling block in parallel computing is an inability to conceptualize parallel approaches to problem solving. People tend to think and solve problems sequentially, but sequential solutions to problems rarely transform into quality parallel solutions. Parallel solutions are said to be *scalable* if additional processors can be used efficiently (66:6). Thus, we are looking for algorithms, such as GAs, that are scalable and exhibit polynomial time complexity.

1.4 Research Objectives

The overall research objective is a synthesis of the search power of generic algorithms and the computational power of parallel/distributed computing that exploits existing and evolving domain knowledge from biochemistry to reliably predict the three dimensional structure of proteins in general, and specifically, modified polypeptides. Said another way–the research objective is the effective and efficient prediction of polypeptide structures based on a triad of real-valued genetic algorithms, biochemistry domain knowledge, and parallel and distributed computing. This "theme" can visualized, Figure 2 as a PSP "table" supported

[5] Crossover is sometimes called recombination.

by three legs, real-valued genetic algorithms, biochemistry domain knowledge, and parallel and distributed computing, resting on a foundation of software and computational engineering principles and experiment design.

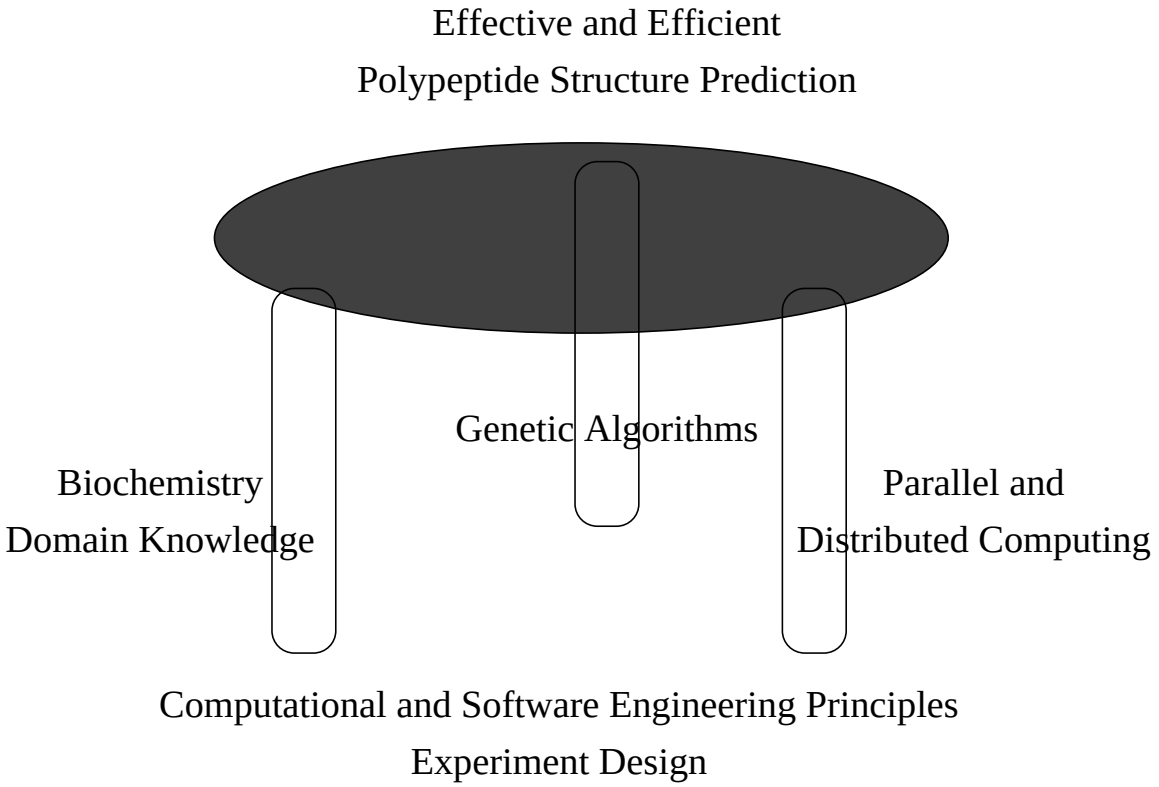


Figure 2. Thesis Theme

Specific objective are the following:

- *Improve Performance of Hybrid GAs for PSP.* The current AGCT hybrid GAs developed by Gates and Gaulke are, to date, the most effective GA for PSP. In addition, the GA community has expended considerable effort defining optimal values for the exogenous parameters in GAs. My objective is to enhance the performance (run time) of these hybrid GAs while maintaining the same behavior (effectiveness).
- *Real Valued Genetic Algorithm Implementation for the PSP.* Current AGCT PFP/PSP GA implementations use binary encoding traditionally seen in GAs. The binary representation has some disadvantages when applied to multidimensional, high-precision numerical problems (84). A real value implementation is designed, implemented, and compared to existing implementations.
- *Exploit Domain Knowledge to Limit Search Space.* The real-valued implementation provides a vehicle to incorporate *domain knowledge* into the GA via linear and non-linear constraints. Initially, values derived from from analysis of existing structures in the Brookhaven National Laboratory's Protein Data Base (PDB) will be used as the basis for constraints.

1.5 Methodology

The existing AGCT hybrid GA for PSP developed in turn by Brinkman, Gates, and Gaulke, is re-engineered as a parallel implementation. It is evaluated against the serial version for correctness (validation) and performance gains. Separately, a steady state real-valued GA is integrated with the AGCT molecular energy and data structure implementations to produce, what Zbigniew Michalewicz calls, an *Evolution Program*[6], This evolution program allows biochemistry researchers to capture and use domain knowledge from their problem space. A systematic study is done of the *exogenous* parameters to characterize their effect in this problem domain. A parallel version is designed and implemented that uses the novel concepts of Probabilistic Migration[7] and Probabilistic Complete Migration.[8]

1.6 Assumptions

It is assumed the AGCT molecular energy models accurately predict the potential energy of the conformer[9] it acts upon. It is also assumed the nomenclature and molecular representations used in the existing AGCT algorithms and data structures are consistent with those published by other researchers in the molecular modeling community. It is also assumed any software developed by this effort is considered "engineering software" (in the acquisition vernacular). As such, certain design alternatives are selected which are not appropriate for "production software". At the same time, one underlying principle of this work, particularly in the parallel and distributed realm, is portability. Therefore, no specific hardware or software implementation is assumed, however, recognized engineering standards are used where appropriate.

1.7 Summary

There exist classes of problems that can't be solved in reasonable time strictly by faster computation. The Protein Folding Problem and the related problem of Polypeptides Structure Prediction are such problems. It has been shown that the PFP and PSP are of intense interest to the scientific community and the USAF. Evolutionary Algorithms, in particular GAs, are used to provide semi-optimal solutions to these problems. This thesis investigation, while continuing the AFIT work in GAs applied the PFP and PSP, takes a radically different approach using real value encoding and search limiting techniques.

This chapter outlines the general problem, describes the main components, and rationalizes the need to expend research effort on genetic algorithms and the protein folding problem. General and specific research

[6] Not to be confused with *Evolutionary Programming*, see Section 2.4.
[7] The probability that an improvement at a specific node is migrated to other nodes.
[8] The probability, given a migration, that it is migrated to all other nodes.
[9] Or molecular conformation

objectives are defined along with assumptions. The next chapter presents salient information in the fields of genetic algorithms, protein structure prediction, and parallel/distributed computation support. The balance of the document is organized as follows: Chapter III discusses algorithm analysis, design, and implementation, Chapter IV discusses experiment design, Chapter V discusses results and analysis, and finally, Chapter VI provides the conclusions and recommendations for subsequent research. Appendix A contains background on protein folding and structure prediction problems. Appendix B contains background on genetic algorithms and other methods of evolutionary computing. Appendix C contains background on parallel and distributed computing. Appendix D contains operation directions for the Parallel Hybrid GA. Appendix E contains analysis of the real-valued GA used in this thesis, GENOCOP-III. Finally, Appendix F is a discussion of the statistical methods used to analyze the results of this research.

II. Current Issues

2.1 Introduction

This chapter provides a review of "current" issues in the disciplines providing the foundations of this research. These areas are Biochemistry, Evolutionary Algorithms (in particular, Genetic Algorithms), and High Performance Computing (in particular, Parallel and Distributed GAs). As a convenience to the reader, a fair base of knowledge in these disciplines is assumed. Thus, introductory material and information is omitted from this chapter. However, for the reader unfamiliar with these disciplines, background material is available in Appendix A, Protein Folding/Structure Predication Problem, Appendix B, Genetic Algorithms, and Appendix C Parallel and Distributed Computing.

2.2 Previous Research

Previous research in GAs at AFIT has resulted in a number of implementations. Collectively these are known as the AGCT Genetic Algorithm Toolkit. Included are general implementations of several serial and parallel genetic algorithms (23, 77) and evaluation functions for several domain specific problems (5, 94, 75). See Figure 3 for the state of the toolkit prior to this research. Figure 31 (Section VI) shows the state of the toolkit after my efforts. Most applicable to this research is the work of Brinkman (initial implementation for PSP) (5), Gates (refinement of energy function, parameter analysis) (35), and Gaukle (integration of local minimization and niching) (36) .

2.3 Polypeptides Structure Prediction (PSP)

The protein folding problem (PFP) is a National Grand Challenge problem in biochemistry and high-performance computing (11). This thesis effort addresses polypeptide structure prediction (PSP), which is one approach to solving the PFP. The challenge of PSP is a method predicting the three-dimensional structure of a protein (or polypeptide) based strictly on its amino acid structure. Much of the nomenclature used in this field was introduced in the previous chapter (Section 1.1). Additional background information is presented in Appendix A.

Kauffman's NK model (64) indicates that, with a high level of epistatic interactions between the variables, the energy surface is extremely multi-modal. Also, his model shows that as the complexity of a landscape increases, basins of attractions rise toward the mean fitness. That is, not only are there many local minimal, there is very little difference between local optima and the global optimal solution (64). Actually, his example was for a maximization problem, hence, the concept for the PSP is inverted. Likewise, Ngo, Marks, and Karplus provide a convincing argument that protein folding via structure prediction is NP-complete (90:433–506).

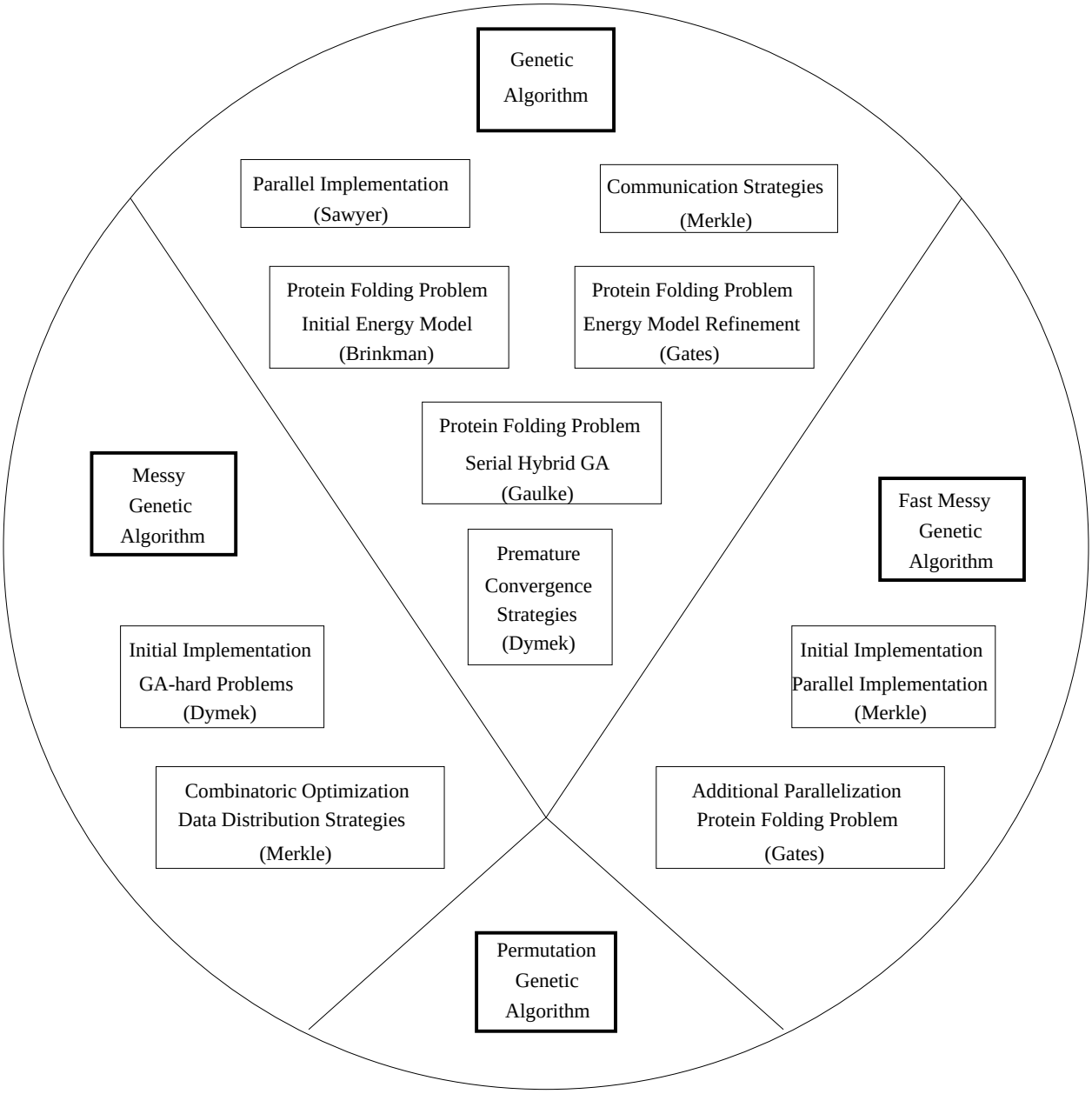


Figure 3. AGCT's Genetic Algorithm Toolkit (Status Prior to This Thesis)

Rabow and Scheraga (98) discuss an interesting GA approach to the PFP using Cartesian coordinates. This paper is interesting for three reasons. First, it marks a shift by Scheraga, historically a Monte Carlo advocate, to GAs. Second, it uses the Cartesian coordinates as variables, encoding them as real values. Finally, the recombination (crossover) operator generates offspring as linear combination permutations of the parents. Scheraga claims this operator, while locally modifying a variable, is less disruptive to the overall conformation. Unfortunately, he uses a simple energy function based on a lattice model. Thus, to evaluate the child, it must be fitted to points on the lattice. This requires large amounts of computation and distorts the candidate solution.

A deterministic method based on the ECEPP energy model using the $\alpha\beta\beta$ algorithm has been presented by Floudas, et al (2). It is guaranteed to produce an $\epsilon - optimal$ solution by computing an upper and lower bound. As the search progresses, the bounds become tighter. This approach uses information mined from commonly available sources, to limit the search space and thus start with a tighter lower bound. $\alpha\beta\beta$ has been applied to the ECEPP/3 model. This technique has promise as a method to identify the optimal CHARMM energy value for Met-enkephalin.

Elsewhere the intractability of the molecular conformation search space has been shown (Section 1.1.4). Ngo, discussing the Levinthal Paradox (90)

> The essence of the (Levinthal) paradox is that in theory a protein is expected to require exponential time to fold, given an arbitrary stating configuration, whereas in practice proteins are observed to fold within seconds to minutes, independent of size.

The argument for exponential folding time is base on the molecule sampling confirmations in a completely random fashion without any "clues" as to the location of its native conformation. The fact is that proteins fold much faster. Experiments indicate the existence of hierarchy in observed protein structures. These suggest that a protein does have clues as to the nature of its native state even when it is quite far from being in the folded state. The clues come from the propensities of parts of the chain to form native-like structures (90).

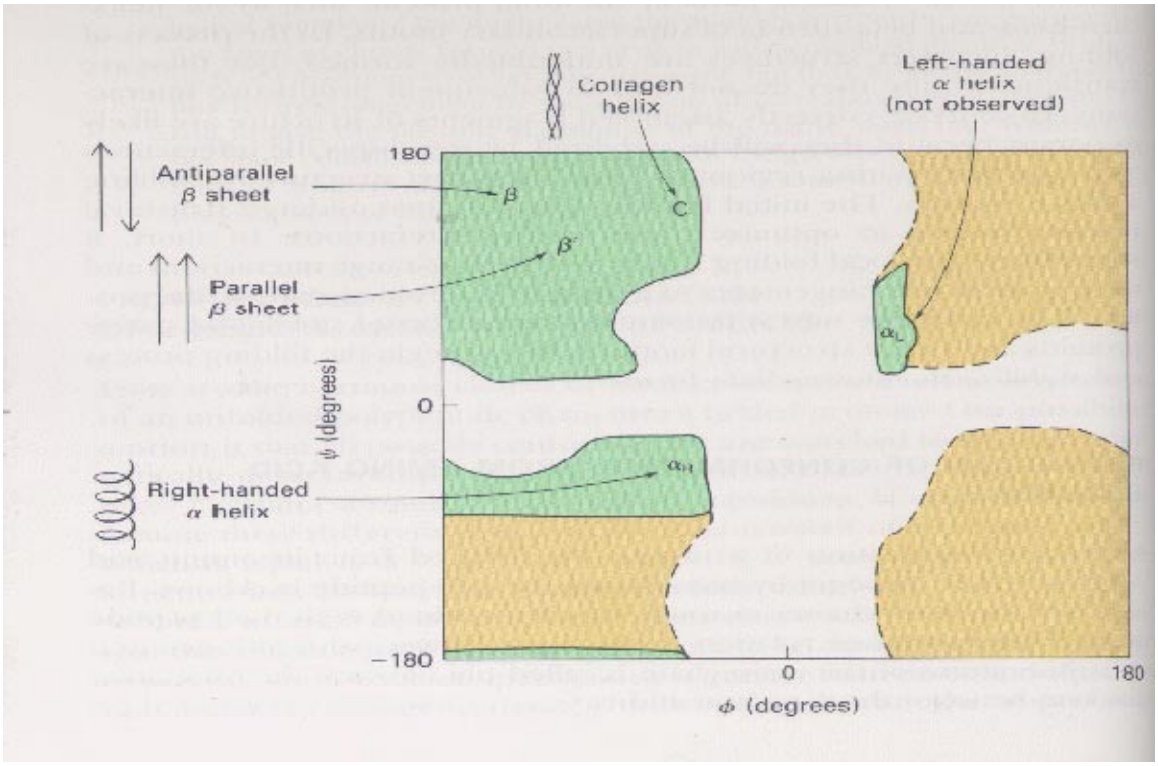
Assuming ω dihedral angle is fixed at the *trans*[1] position, the (ϕ, ψ) pair of dihedral angles defines the backbone of a polypeptide. Many times researches use the backbone as a more tractable abstraction (or simplification) of the molecular system under study. In a large molecule the backbone dihedral angles (ϕ, ψ, ω) are responsible for defining the correct folded state. However, it seems that the side-chain dihedral angles $(\chi's)$ play an important role in determining the energetically most stable configuration (2).

In the early 1960's, Using hard-sphere models of the atoms and fixed-geometries of the bonds, Ramachandran and colleagues (14) derived regions in terms of the allowed values of the (ϕ, ψ) dihedral angles.

[1] $\pm 180^{circle}$ or $\pm \pi$

The key result of their calculations was that for every naturally occurring amino-acid, similar patterns were observed.

The *Ramachandran Plot*, see Figure 4, is a tool widely used in the molecular modeling. It plots ϕ versus ψ values, for anywhere from a single residue up-to thousands of angles. Scheraga identified similar patterns using a build up process where he generated thousand of conformations for the 20 amino acid resides. These conformations where then evaluated using the ECEPP energy model to identify those most likely to occur in nature (2).



From *Biochemistry* by Stryer. Data derived from Brookhaven Protein Data Bank.

Figure 4. Ramachandran Plot

The official repository of protein structures in the United States is the Brookhaven Protein Data Base (PDB) operated by the National Institute of Health. Access to the database is available via the World Wide Web. The Dictionary of Protein Secondary Structures (DSPP), hosted in Germany, is another excellent source for data about molecular structure that have already been identified. In DSPP, dihedral angle values are readily available, where as the PDB files from Brookhaven use Cartesian coordinates.

2.4 Genetic Algorithms

The focus of this section is on data representation issues and the concept proposed of an *evolution program*. See Appendix B for comprehensive information on GAs.

In his paper's introduction at the first Parallel Problem Solving from Nature (PPSN) conference (39), David Goldberg captures the essence of the GA data representation issue:

> The use of real-coded or floating-point genes has a long, if controversial, history in artificial genetic and evolutionary search schemes, an there use as of late seems to be on the rise. This rising usage has been somewhat surprising to researchers familiar with genetic algorithm (GA) theory(Goldberg, 1989;Holland, 1975), because simple analyses seem to suggest that enhanced schema processing is obtained by using alphabets of low cardinality, a seemingly direct contradiction of empirical findings that real codings have worked well in a number of practical problems. The debate between practitioner and theoretician over this *paradox of real codings* has risen almost to the point of schism. Theoreticians have wondered why practitioners have paid so little heed to the theory, and practitioners have wondered why the theory seems so unable to come to terms with their findings.

Goldberg attempts explain this paradox by the notion of *virtual alphabets*. That is, a real value coding forms "pools" that behave as individual characters in a virtual alphabet. It also suggest that the real-coded GAs can be *blocked* from further progress in those situations when local optima separate the virtual characters from the global optimum. On the other hand, Wright (113) shows that Holland's schema theorem, the *Fundamental Theorem of Genetic Algorithms*, holds for real-valued codings by viewing an interval on the number line as a schema.

The concept of an evolution program is used throughout this research effort. The concept, and choice of nomenclature, are best described by Michalewicz

> I used the term "Evolution Programs" to mean some generalization of genetic algorithms. The book was written in 1991, when (apart from GAs) other algorithms were less known. Also, I tried to make a parallel to the title of famous book by N. Wirth: "Algorithms + Data Structures = Programs". His title suggests that to construct a successful program, appropriate algorithm should be used with appropriate data structures. The same is with evolutionary stuff: appropriate algorithms (in terms of problem-specific operators) plus appropriate data structures (as chromosomal representation).(83)

In general, AI problem solving strategies are categorized into *strong* and *weak* methods (84). A weak method makes few assumptions about the problem domain; hence it usually enjoys wide applicability. On the other hand, it can suffer from combinatorially explosive solution cost when scaled up to larger problems. This can be avoided by making strong assumptions about the problem solving method. But a disadvantage of such strong methods is their limited applicability. Figure 5 shows a series of evolution programs EP_1 through EP_5. Starting with EP_1, which is the weakest, they become more problem specific until EP_5, the strongest, is applicable to only the problem P (79).

Figure 5. Hierarchy of Evolution Programs, from (Michalewicz 1993)

The trade off between efficiency and problem spectrum (broad applicability) is shown in Figure 6. Notice how the efficiency increases when the problem spectrum is limited. For example, Q could be a commercial package designed to deterministicly solve problem P. EP_1 represents a classical GA, while EP_5 represents a very focused evolution program.

2.5 Parallel Genetic Algorithms

General information about parallel computing is available in Appendix C. There are two major concerns when parallelizing any algorithm: is the parallel algorithm correct (effective) and is it faster (efficient) than the serial version? Correctness is an issue because there is greater difficulty in verifying parallel algorithms than sequential programs (72). Given that the parallel algorithm is correct, speedup is the primary goal of parallelization (9). A tradeoff analysis is generally required to determine if the estimated benefits warrant the expenditure of resources to parallelize an algorithm. There is evidence to suggest that parallelizing genetic algorithms *is* worthwhile and should be examined further (22, 48, 96, 104, 109). Parallel GAs can be synchronous or asynchronous, which in some cases actually improves performance (54).

Figure 6. Efficiency/Problem Spectrum and Evolution Programs, from (Michalewicz 1993)

Data and control decomposition are alternate means of dividing a problem into portions that can be worked on simultaneously. In general, data decomposed algorithms perform the same operations on subsets of the input (*data parallelism*) and control decomposed algorithms perform different operations on the total input (73). In either case the results are recombined in a fashion to obtain the final result(s).

Genetic algorithms are easily parallelized because they are highly data decomposable (although control decomposition is not impossible, especially as the complexity of operators and evaluation functions grow). Parallelizing GAs using data decomposition can be as simple as running multiple copies of the same program on different processors, each starting with a different random number seed, and then choosing the best result from all runs.

Data parallelization techniques are also amenable to static load balancing because their computation and communication patterns are regular (24, 73). This does not imply that all processors are searching in equally promising portions of the search space. Thus, some efficiency may be lost to subpopulations that are searching similar solution neighborhoods or stuck in local optima.

Two models, which lie at opposite ends of a granularity spectrum, have been proposed for parallel genetic algorithms—the *island model* (course-grained) and the *neighborhood model* (fine-grained) (21, 48). These models are designed to improve the simplistic parallel approach by sharing near-optimal results with some portion of the global population. The *farming model*, on the other hand, decomposes control, rather than data. All three are discussed in the following sections.

Island Model: The island model is an extension of the simplistic approach where the total population is divided into subpopulations which are distributed among the processors. The subpopulations evolve in parallel, however at certain time intervals a *migration* occurs where solutions are communicated between processors (21:10). Migration rates, migration selection strategies, and migration patterns are additional parameters with associated design decisions that must be defined for the parallel genetic algorithm. Near-linear speedup is expected and has been observed for island model parallel genetic algorithms (5:60) (7, 109). The time complexity of island model GAs is $\mathcal{O}(\frac{nl}{p})$, where p is the number of processors, n is the population size, and $p \ll n$ (21). As $p \rightarrow n$, the resulting small subpopulation size increases the ratio of communication time to compute time and the speedup becomes much less than linear. Island model GAs are typically used on course-grained or multiple-instruction-multiple-data (MIMD) architectures (65).

Neighborhood Model: The neighborhood model splits the population up spatially in a two- or three-dimensional grid. This grid and the definition of a neighborhood limits the interaction of individuals in the population. Typically, a single string is assigned to each processor, therefore crossover and selection must be modified because their operation is distributed across more than one processor. Although their convergence characteristics have been observed to be better than the Island Model (21:24), parallel genetic algorithms of the neighborhood model don't exhibit speedup because they assume $n = p$, therefore no speedup can be obtained because more/fewer processors are not allowed. The neighborhood model is most often compared to the simple GA for time complexity analysis ($\mathcal{O}(s+l)$ vs $\mathcal{O}(nl)$ where s is the neighborhood size) (21:24). Neighborhood model GAs are generally implemented on fine-grained or single-instruction-multiple-data (SIMD) architectures (21).

Farming Model: The term *farming* is not used in an agriculture context. Rather, it is used in a manufacturing context, as in to *farm out* work (102), in this case, complex computations. It consist of one entity called a *foreman* and one or more *workers*. The foreman makes the decisions, including assigning work. The workers perform the assigned task. As in the real world, the workers are sometimes idle, awaiting work from the foreman. One advantage to this approach is that its effectiveness is identical to the serial GA. Thus every thing known about the behavior of the serial algorithm, both theoretical and empirical, can be exploited.

2.6 Summary

The determination of the tertiary structure of proteins is a major challenge in biochemistry. Experimental techniques are considered accurate but time consuming, and are incapable of keeping pace with the number of protein sequences being discovered. Prediction techniques are hampered by the size of the conformational search space and the time complexity of calculating energy or solving motion equations.

However, classical prediction methods, combined with novel search and optimization algorithms, show great potential for both a solution to the PFP and a better understanding of the underlying behavior and operation of biological systems. This thesis effort considers the application of real-valued genetic algorithms using energy minimization as one such combination for solving the protein folding problem. In particular, the use of domain knowledge to limit the search (solution) space and thus increase the efficiency is investigated. The next chapter analyzes issues from the problem and algorithm domains, and discusses the design and implementations of three refined GAs.

III. Algorithm Analysis, Design, and Implementation

The previous chapter reviewed the current literature in the three domains that support this research; Biochemistry, Evolutionary Algorithms, and High Performance Computing. This chapter is divided into two parts. The first half analyzes specific issues from the above domains. The second half explains design and implementation details for three refined GA algorithms.

3.1 Analysis

3.1.1 Cost Analysis of Local Minimization using Conjugate Gradient. This section analyzes the conjugate gradient technique (CG) for local minimization with respect to PSP. The objective is to identify a strategy for overcoming the increased run time cost observed in the serial implementation.

A CG minimization of the CHARMM energy model was adapted from *Numerical Recipes in C* (97) by Gates and Gaulke (36, 78). While a hybrid GA with CG minimization is more effective for PSP than a SGA, it incurs a substantial run time penalty. Analysis of the algorithm is not trivial. CG is an iterative method and converges in n or fewer iterations for a quadratic function (66) where n is the number of variables. However, the CHARMM energy function (Equation 16, Appendix A.3) is far from quadratic. Every iteration involves the cost of a call to the fitness function, a $\mathcal{O}(n^2)$ operation. A constant (`ITMAX = 200`) is used as an upper bound on the number of iterations attempted. Empirical results of profiling the serial hybrid GA developed by Gates and Gaulke are presented in Table 1. The profiling represent equivalent runs of 500 evaluations against a 20 member population of the molecule Met-enkephalin on similar workstations (Sun Sparc 2's). *Lamarckian* and *Baldwinian* minimization techniques are used for instances of CG, while a simple GA is used as a baseline. The two orders of magnitude difference in time per call to the fitness evaluation function, `eval_func()`, between the CG minimized and non-minimized GAs suggest that, frequently, the number of iterations per local minimization reaches the upper bound, `ITMAX`.

The Lamarckian approach represents the concept of learning from the environment which is passed on to the offsprings via replacing the chromosome's genotype (molecular structure) as well as the phenotype (fitness) resulting from minimized chromosome (111). The Baldwinian approach also represents the concept of learning from the environment but with out passing the knowledge on to offspring. Here only the phenotype (fitness) resulting from minimized chromosome is used in the selection process. Baldwinian can also be thought of as identifying a a favorable region of the search space, which is given an increased chance of selection by association with the fitness resulting from minimization (111).

The hybrid GA under study here, like most classical GAs, is *generational*. That is, it follows the algorithm presented in Appendix B.3. Thus for every generation there is a *checkpoint* at which *asynchronous* processes could rendezvous. This fits into a *farming* of parallel computation. Here, farming is used in

manufacturing rather than agriculture sense. A supervisory node, called the *client*, distributes (farms) work to one or more worker nodes, called a *server*. When the server completes the work, the results are returned to the client. In order for this approach to be successful, the computational effort involved must be significant compared to the communications cost (a coarse grained algorithm). Thus the need to identify large chunks of computation in the algorithm, which can be accomplished independent of the other chunks, but doesn't require large quantities of dynamic data. Static data can be replicated *a priori* at the server nodes.

The three classic operators (selection, crossover, and mutation), and the evaluation operation, are examined to identify candidate processes to be farmed out. The selection operator, especially when using fitness proportion selection, is a synchronous process and requires access to the entire population; thus, is not a candidate. The crossover is a binary operator requiring access to two chromosomes at a time. In this application each chromosome is operated on by crossover only once per generation. While it is an asynchronous process, it requires negligible computation time. Total time for crossover in the case study was less than one second, a poor candidate for farming. Mutation is an independent unary operator and thus an asynchronous process. However, it has the same problem as crossover but worst. While the code profiler records the function calls to crossover, the times are so small they fail to register. The evaluation operator is also an independent unary operator and thus an asynchronous process. This operator is the perfect candidate for farming as its computation dominates the GA, requiring approximately 80% of the total computation time for the simple GA and over 99% for the GAs using CG, see Table 1.

Table 1. Serial Hybrid GA Profiling Results

Minimization Algorithm	Total Exec Time (sec)	Time in charm_eval()	% Time in charm_eval()	Calls to charm_eval()	Avg Time/ Call (sec)
Lamarckian	4989.55	4978.78	99.78	507	9.84
Baldwinian	4030.19	4011.4	99.53	504	7.96
Simple	56.67	46.11	81.37	514	0.09

The evaluation operator is implemented for PSP as function `charm_eval()`. Since the CG minimization is specific to the function being minimized, in this case the CHARMM energy function, it is hidden, in the software engineering sense, behind `charm_eval()`. Not surprising, the test cases confirm that the increase in run time involving CG local minimization involves `charm_eval()` or one of its children (which includes the CG functions). There is a two orders of magnitude increase in in both the total execution time, and time per call to `charm_eval()`, when local CG minimization is used (Lamarckian or Baldwinian).

Finally, the software design supports this as an ideal point from which to "farm" out work. The algorithm domain, instantiated by the modified version of the GENESIS software, is very loosely coupled from the application domain, instantiated by the CHARMM energy function and related molecular struc-

ture transformation functions. The implementation of the Parallel Hybrid GA (PHGA) is discussed in Section 3.2.1.

3.1.2 Constraint Set Development. My focus here is on development of techniques usable by biochemistry researchers for structure prediction. The constraint sets developed in this research demonstrate the feasibility of the approach. In general they are conservation, that is, less restrictive. A biochemistry researcher studying a particular molecule may choose to develop more aggressive constraints. This flexibility is inherent in my design.

3.1.2.1 Conventions Adopted. The convention of the chemistry community is for dihedral angles to range from $-180°$ to $180°$. Also, $0°$ is at the top of a unit circle, while $-180°$/ $180°$ is at the bottom. The numbers increase in a clock wise direction. To consistently apply the domain constraints the following convention is defined.

> When defining a constraint, values for the lower and upper bounds of the valid region, θ_{min} and θ_{max} must be assigned such that one travels from θ_{min} to θ_{max} in a clockwise direction.

This creates a pathological condition when the the valid region wraps (involves the $-180°$/ $180°$ position). For the generalized constraint forms, Equations 6 and 7, discussed in Section 3.2.2.1, to work properly, θ_{min} to θ_{max} are adjusted in the following manner.

$$\theta_{min} > \theta_{max} \rightarrow \theta_{min_{adj}} = \theta_{min} - 360° \tag{2}$$

$$\theta_{min} < \theta_{max} \rightarrow \theta_{max_{adj}} = \theta_{max} + 360° \tag{3}$$

The resulting $\theta_{min_{adj}}$ and $\theta_{max_{adj}}$ may be outside the range $-180°$ to $180°$. However, Equations 6 and 7 function correctly because the phase shift component is cyclical while the difference will be positive as required by the scaling component. These adjustment have been made to θ_{min} and θ_{max} in Tables 2–5.

3.1.2.2 [Met]-enkephalin. The "loose" constraints for [Met]-enkephalin (Table 2) were developed by examining Ramachandran plots of observed values of *phi* and *psi* angle for the residues alanine and glycine (13). Of the twenty amino acids, proline and glycine have unique $\phi\psi$ distributions. The other residues are similar to alanine. The "tight" constraints (Table 3) consider the above data and infer additional insights from "homologous" molecules.

3.1.2.3 Polyalanine. Values for the Polyalanine constraints (Table 4) were developed in a similar way. It was known *a priori* that this molecule forms an α-helix secondary structure. Thus a plot from Stryer's text (108) that specifies the $\phi\psi$ region for an α-helix was used. A similar process to that above was

Table 2. Loose constraints for [Met]-enkephalin

Dihedral	Midpoint	Radius	θ_{min}	θ_{max}
$\Phi_{Non-glycine}$	-120	90	$150 - 360$	-30
$\Phi_{Glycine}$	-180	135	$45 - 360$	-45
Ψ	60	150	-90	$-150 + 360$
Ω	-180	20	$160 - 360$	-160
χ_1	$-60 \mid 60 \mid 180$	30	$-75 \mid 45 \mid 175 - 360$	$-45 \mid 75 \mid -165$

Table 3. Tight constraints for [Met]-enkephalin

Dihedral	Midpoint	Radius	θ_{min}	θ_{max}
$\Phi_{Non-glycine}$	-120	60	-180	-60
$\Phi_{Glycine}$	130	70	$60 - 360$	-160
Ψ	150	140	$10 - 360$	-70
Ω	180	12.5	$167.5 - 360$	-167.5
χ_1	$-60 \mid 60 \mid 180$	7.5	$-67.5 \mid 52.5 \mid 172.5 - 360$	$-52.5 \mid 67.5 \mid -172.5$

used for the "tight" constraints (Table 5). After consulting with biochemistry domain experts, a third set of constraints "tight, relaxed terminals" were defined. These are based on the knowledge that the dihedral angles for the terminals residues will not be consistent with the non-terminal angles even in a very regular secondary structure like an α-helix. They are the same as the "tight" constraints, but without constraints on residues 1 and 14.

Table 4. Tight constraints for Polyalanine

Dihedral	Midpoint	Radius	θ_{min}	θ_{max}
Φ	-67.5	22.5	-90	-45
Ψ	-30	30	-60	0
Ω	180	20	$160 - 360$	-160
χ_1	$-60 \mid 60 \mid 180$	30	$-90 \mid 30 \mid 150 - 360$	$-30 \mid 90 \mid -150$

3.1.3 Real-valued GAs. As indicated elsewhere, a principle objective of this research is to evaluation the performance that can be obtained using a real-valued GA for PSP. To accurately assess the difference between binary and real-valued data representation, the two implementations ideally should identical. The GENESIS system, which is the basis for other implementations in the AGCT tool box, such has the hybrid GA discussed in Section 3.1.1, provides a real-valued interface. However, this is just a convenience for interfacing to a real-valued fitness function. The genes are actually converted to a binary representation, which is acted upon by traditional binary-valued operators. As indicated in the literature search, the greatest benefit from real-valued GAs is the availability of richer operators. Of the real-valued GAs available in the public domain,

Table 5. Tight constraints for Polyalanine

Dihedral	Midpoint	Radius	θ_{min}	θ_{max}
Φ	-60	15	-75	-45
Ψ	-45	15	-60	-30
Ω	180	5	$175 - 360$	-175
χ_1	$-60 \mid 60 \mid 180$	5	$-65 \mid 55 \mid 175 - 360$	$-55 \mid 65175$

GENOCOP-III was selected because of its use for non-linear constraint problems. GENOCOP-III[1] is the new version of this system for handling Numerical Optimization of Problems with Linear and Non-Linear Constraints. It is been used as the real-valued GA

3.2 Algorithm Design and Implementation

This thesis effort involves three distinct GA algorithm designs and implementations. The first, discussed in Section 3.2.1, is a parallel implementation of the existing AGCT hybrid GA. The second, discussed in Section 3.2.2 is a real-valued implementation mating the molecular model with the GENOCOP-III package. The third, discussed in Section 3.2.3 is a parallel implementation of the second algorithm.

3.2.1 Parallel Hybrid GA. The Parallel Hybrid GA (PHGA) is implemented by modifying the serial hybrid GA developed by Brinkman (5), Gates (35), and Gaulke (36) into a *farming model* parallel GA. Message passing is accomplished via the Message Passing Interface (MPI) library (101).

3.2.1.1 Algorithm Design. Upon initialization, each node determines its role, either client or server. In either role, the node performs domain initialization, building a molecular model of the polypeptide to be studied. The client generates an initial population and proceeds as though it were a serial algorithm, except its fitness evaluation (and optionally, local minimization) tasks are "farmed" out to a server via a synchronous send message. The server computes and returns the fitness and an updated chromosome to the client via a synchronous reply message. To maintain the same evolutionary trajectory as the serial execution, the client synchronous each generation. That is, it does not proceed to the next generation, G_{t+1}, until all evaluation task farmed out in generation, G_t, have been returned. When the required number of evaluations are accomplished, or some other termination condition is reached, the client broadcast an asynchronous termination message to the servers. The client then performs its normal domain output and other termination activities. Pseudo-code for the client node is in Figure 7. Server node pseudo-code is in Figure 8 and pseudo-code for the farming operation is in Figure 9.

[1] The developers of this algorithm have been inconsistent with its naming in the literature. Genocop, Genocop-III.1.0, GENOCOP-III, etc., have been used synonymously. In this document I have adopted the standardization of GENOCOP-III.

1. Build molecular model
2. Randomly generate initial population
3. Put servers in Q
4. $EvalCnt \leftarrow$ *population size*
5. **Loop**
6. Perform farming operation
7. **Until** *EvalCnt* evaluations performed
8. **Loop** – Main Body
9. Perform selection
10. Perform crossover
11. Perform mutation
12. $EvalCnt \leftarrow$ evaluations needed in this generation
13. **Loop**
14. Evaluate by perform farming operation
15. **Until** *EvalCnt* evaluations performed
16. **Until** maximum number of evaluations or other stopping condition
17. Broadcast termination message to servers
18. Perform domain wrapup

Figure 7. Farming Model for PSP, Client node

3.2.1.2 Scheduling. Scheduling involves the distribution of work to available servers. The objective is to keep all nodes equally busy. Initially, scheduling was via a *round robin* scheduler. When a server was needed to perform a task, the processor with the Node_ID one greater than that previously used was selected. When the highest number Node_ID had been used, we wrapped around back to Node_ID 1, thus bypassing the client node, which is always Node_ID 0. In theory, the round robin technique should be very equitable. The reality is the severs do not always return work in the order dispatched. This is especially true on a *network of workstations* (NOW) where unbalanced system loadings resulting from the multi-user environment and network conflicts cause large variability in individual performance.

The revised scheduling technique is a First In, First Out (FIFO) queue. The queue is adapted from the classical circular queue (array implementation) found in basic data structures textbooks. During initialization, the client creates a queue and enqueues IDs of all server nodes. When a tasked is to be farmed out, the server at the front of the queue is used. When a server returns it task, its ID is put into the back of the queue. The revised scheduler yielded a slight, but measurable, decrease in run time.

3.2.2 REal-valved GA, Limited by constraints (REGAL) . REGAL is my principle contribution to the AGCT Tool box (Section 2.2). A genetic algorithm, GENOCOP-III, is integrated with data structures, i.e. constrained real-valued variables and other domain constraints, to form, what Michalewicz calls, an *evolution program*, (see Section 2.4).

```
1.  Build molecular model
2.  Loop
3.      Receive message from client
4.      If MINIMIZE
5.          Perform local minimization (implicit fitness evaluation)
6.          Update fitness of individual in message buffer
7.          If REPLACE
8.              Replace message buffer string with minimized string
9.      Else
10.         Compute fitness, update message buffer
11.     Send message buffer to client
12. Until termination message received
```

Figure 8. Farming Model for PSP, Server node

```
1.  If reply message waiting
2.      Update fitness of individual with value from message
3.      If REPLACE
4.          Replace chromosome in population with minimized string from message
5.      Enqueue server_i in Q
6.      Increment EvalsPerformed
7.  If EvalsPerformed < EvalCnt ∧ Q ≠ ∅
8.      Dequeue server_j from Q
9.      Send tasking message containing string and local min parameters to server_j
```

Figure 9. Farming Model for PSP, Farming Operation

3.2.2.1 Incorporation of Domain Knowledge. As discussed in previous chapters, there is a growing body of knowledge applicable to PSP. Likewise, there is little dispute that a "strong" algorithm out performs a "weak" algorithm in its domain. This is the premise of Michalewicz's *Evolution Program* (see Section 2.4). But how do we incorporate knowledge from the problem domain into REGAL? The following approach is perhaps the greatest contribution to the PSP from this research effort.

The analysis in Section 3.1.2 yielded a number of constraint sets. The constraint sets, defined over the independent dihedral angles, divide the search space into *probable* and *improbable* regions. Let $\mathcal{S}_{prob}$ represent the probable search space and $\mathcal{S}_{improb}$ represent the improbable search space. Thus formally,

$$\mathcal{S}_{prob} \bigcup \mathcal{S}_{improb} = \mathcal{S} \tag{4}$$

$$\mathcal{S}_{prob} \bigcap \mathcal{S}_{improb} = \emptyset \tag{5}$$

A constraint set defines a $\theta_{i_{min}}$ and $\theta_{i_{max}}$ for each $x_i \in \vec{x} \in \boldsymbol{R}$ that is active in the constraint set. If x_i is not active in the constraint, $\theta_{i_{min}}$ and $\theta_{i_{max}}$ default to $-\pi$ and π repectively. The constraint set can be transformed into a series of nonlinear inequalities of one of the two following forms:

$$0 \leq \cos(\theta - \frac{\theta_{min} + \theta_{max}}{2}) - \cos(\frac{\theta_{max} - \theta_{min}}{2}) \tag{6}$$

are the constraints for the $\{\phi, \psi, \omega\}$ angles, and

$$0 \leq \cos(3\theta - \frac{\theta_{min} + \theta_{max}}{2}) - \cos(\frac{\theta_{max} - \theta_{min}}{2}) \tag{7}$$

are the constraints for the $\{\chi_1\}$ angles which have a trimodal nature similar to that in Figure 1.

The nonlinear inequalities are hard coded into the software prior to compilation. Each set is linked to a particular case identifier. The specific set to be applied is indicated by a parameter supplied at runtime in the input file.

3.2.3 Parallel REGAL (Para-REGAL). Para-REGAL is an *modified island* model parallel implementation of REGAL for PSP. The traditionally the parallel GA *island* model evolves the subpopulation at a particular node in an environment identical[2] to the environment at all other nodes. At some interval, a portion of the subpopulation is migrated to other nodes. Para-REGAL modifies this model in two ways. First, in Para-REGAL, each node can be an unique environment. In addition to unique seeds, different active constraint sets, different exogenous parameters, and different explicit initial populations can be specified. The second modifications involves a *probabilistic migration* policy. Two parameters, *Probability of Migration* (P_m) and *Probability of Complete Migration* (P_{cm}). An update to the local reference population is the trigger for a migration which is accomplished with probability P_m. P_{cm} is the probability the migration reaches all other non-terminated nodes. Thus the probability node i will receive the update to node j's reference population is $P_m \cdot P_{cm}$.

Assumming $P_m \cdot P_{cm} > 0.0$, the migration of fit individuals between islands provides an opportunity for each island to maintain a sense of progress across the *archipelago*.[3] At a specified interval, the island takes a checkpoint computing the following metrics:

- *Average Genotypic Distance.* Arithmrtic mean, with respect to the local best chromosome, of genotypic distances to the best chromosome at each non-local island.
- *Least Genotypic Distance.* Distance to the genotypicly nearest island.

[2] Except for the seed of the random number generator.
[3] A collection of islands; thus the islands making of the Para-REGAL execution

- *Greatest Genotypic Distance.* Distance to the genotypicly farthest island.
- *Average Best Fitness.* Arithmetic mean of the best fitness values.
- *Local Delta.* Differnce between local best fitness and the average best fitness.

Experiments to evaluate Para-REGAL are discussed in Section 4.5 and results are presented in Section 5.4.

3.3 Summary

This chapter has examined three issues from the problem and algorithm domains. A hybrid GA using conjugate gradient local minimization was analyzed in Section 3.1.1 to increase performance by farming out computationally intensive tasks in a parallel GA. The design of the resulting refined GA was presented in Section 3.2.1. Section 3.1.2 analyzed data available from the biochemistry domain to develop five sets of constraints for the two test molecules used in this research, [Met]-enkephalin and Polyalanine. In addition, the choice GENOCOP-III as the basis for subsequent real-valued GA research was presented in Section 3.1.3. From these, two additional refined GAs, REGAL, and Para-REGAL, were derived. Design details were discussed in Sections 3.2.2 and 3.2.3. The next chapter presents experiments design to evaluated these refined GAs.

IV. Experiment Design

From the previous chapters, the hypothesis has emerged that a real-valued GAs could be augmented with knowledge from the problem domain to form a more powerful, if limited, search method. Will it work? Is it flexible? Are there any unforeseen issues? How does this approach compare with speeding up an existing good, but slow, method by distributing the workload in a high performance computing environment? What's the benefit? What are the limitations? The experiments described in this chapter are intended to answer these questions.

General experimental issues are discussed in Section 4.1. Experiment I, Section 4.2, evaluates a parallel implementation of the AGCT Hybrid GA. Experiment II, Section 4.3, evaluates the feasibility of exploiting domain constraints with respect to PSP. Experiment III, Section 4.4, is specifically designed to characterize the effect of exogenous input parameters with respect to REGAL because of issued raised during Experiment II. Experiment IV, Section 4.5, is an analysis of a constraints and probabilistic migration applied to parallel and/or distributed real-valued GA implementation for PSP.

4.1 Experiment Techniques

The following two sections provide background on experiment designs issues. In particular, the methods used to generate random number generator seeds, Section 4.1.1, and statistical analysis methods, Section 4.1.2, are discussed.

4.1.1 Random Number Seeds. The stochastic nature of GAs is totally dependent on the random number generator. Dymek's thesis (23) studies the random number generator used in Genesis. He shows that the sequence generated does in deed appear to be random. However, he raises an concern about the algorithm used by Sawyer (67) to generate *nodal* seeds. A nodal seed is the specific seed for a node in a multi-nodal parallel algorithm. The algorithm used by Sawyer is:

```
Seed =(unsigned) ((Seed + My_node) / (My_node + 1);
```

As Dymek points out, the integer division results in this being a step function. For small seed values (and larger node counts), this result in many nodes having the same seed value. Obviously, if seeded identically, the pseudo-random numbers that result induce identical behavior in each like seeded node–a waste of resources.

To over come the above problem, three separate approaches are taken. First, where diversity is intended, the seeds are randomly generated. The procedures for Genesis based, and GENOCOP based, GA software are discussed in the next paragraph. The second approach was to modify the nodal seed generating algorithm by replacing the integer division with a modulus operation:

```
Seed =(unsigned) ((Seed + My_node) % Max_Seed_Value);
```

The third approach, used in the client-server farming node, was to have the client generate all random numbers and pass the random number to the server. This resulted in almost identical results between the serial and parallel algorithms. Thus Dymek's desire for a parallel GA that "performs as close as possible to the sequential version, only faster (23:154)" is accomplished.

The Genesis GA engine requires a single nine digit seed value. The five values for Genesis in Table 6 are generated with the following algorithm:

$$Seed = floor((999999999 * rand\ number\ from\ Xcalculator) + 1) \tag{8}$$

The GENOCOP III GA engine requires two random number seeds. The Seed1 ranges from 0 to 31328. The Seed2 ranges from 0 to 30081. The five seed pairs in Table 6 are generated using the following algorithm:

$$Seed\ 1 = floor((31328 * randon\ number\ from\ Xcalculator) + 1) \tag{9}$$

$$Seed\ 2 = floor((30081 * randon\ number\ from\ Xcalculator) + 1) \tag{10}$$

Table 6. Random Number Seeds

Set	Genesis Seed	GENOCOP Seed 1	GENOCOP Seed 2
1	712874273	26482	13328
2	245786357	18695	06208
3	840511331	25743	22696
4	924771098	05695	11275
5	101185467	11816	22450

4.1.2 Statistical Techniques. This thesis presents the results of a number of experiments in which various implementations of genetic algorithms are compared. In order to draw meaningful conclusions from the results of those experiments, statistical hypothesis testing is required (1:483). Analysis of Variance and the Kruskal-Wallis H test are used for hypothesis testing. Details of these is in Appendix F.

4.1.2.1 Analysis of Variance (ANOVA). Suppose we have a treatments or different levels of factors we wish to compare. Each of the n observed responses from each of the a treatments is a random variable. For hypothesis testing, the model errors are assumed to be normally independently distributed random variables with mean zero and variance of σ^2. The test is to determine if the means of two (or more) separate populations (results at various treatment levels) are the same. ANOVA can be used to compare a

number of population means, simultaneously, thus avoiding the need for a large number of two-sample tests (1).

4.1.2.2 Kruskal-Wallis H Test. Many statistical tests are applicable only when the data may be assumed to be normally distributed. Such tests are not appropriate to some experiments in this thesis. The populations in these experiments cannot be assumed to be normally distributed because there is a known bound on the experimental data. The existence of a bound on the experimental data is inconsistent with the assumption of a normal distribution. Furthermore, in most cases the bounds are near the observed means of the experimental data, so that the errors introduced by the assumption of a normal distribution are likely to be quite large. For these experiments, Kruskal-Wallis H Test is used.

4.2 Experiment I: Evaluation of the Efficiency of a Parallel & Distributed Hybrid GA

4.2.1 Motivation and Objective. The conjugate gradient local minimization technique used in the hybrid GA is in terms of effectiveness, superior to the simple GA (36, 78). However, this is at a substantial cost in terms of efficiency as seen in Section 3.1.1. Since substantial effort has been invested in identify effective parameters for simple GAs (35, 37), it is desirable to maintain behavior that results in the same solution, only arriving there much quicker.

The objective of this experiment set are twofold. First, validate that the results obtained from the parallel implementation are equivalent to those obtained by the serial implementation. Secondly, characterize empirically the efficiency of the parallel implementation in terms of overhead, speed-up, and scalability.

4.2.2 Methodology. Research previously conducted by Gaukle (36, 78) was analyzed to determine which minimization techniques produced the best results. Test case based on the selected techniques are shown in Table 7. The probability of minimization is given by P_m. Likewise, the probability of replacement is given by P_r.

Experiments for each test case are run using 1, 2, 6, 12, 18, and 24 processors on the Aeronautical System Center's (ACS) Major Shared Resource Center's (MSRC) Intel Paragon. The serial implementation developed by Brinkman, Gates, and Gaulke (5, 35, 36) was used for the single mode experiment.

A *farming model* parallel implementation based on the serial hybrid GA was used for the multi-node experiments. Implementation details of are given in Section 3.2.1. Comparison between the single node serial runs and the 2 node parallel runs was used to measure the overhead imposed by the communications library. Parallel runs on 6, 12, 18, and 24 processors are selected to evaluate the speed-up and scalability

of this algorithm. 24 nodes was selected as an upper bound since it exceeds the population size of 20 which historically has produced the best results (35).

Table 7. Hybrid GA Test Cases

Case	Selection	Minimization Technique	P_m	P_r
A	Fitness Proportional	Simple	0.0	0.0
B	Tournament Selection	Simple	0.0	0.0
C	Fitness Proportional	Baldwinian	1.0	0.0
E	Fitness Proportional	Lamarckian	1.0	1.0
F	Fitness Proportional	Lamarckian	1.0	1.0

An objective of this experiment set is validation of equivalent results from the serial and parallel implementations. Therefore, all runs used the same random seed value `987654321`. Runs used population sizes of 20, 50, and 100, for 500, 1000, 1500, and 2000 evaluations. Three replicates of each cell are performed. The result is 1440 individual experiments! (240 with the serial algorithm and 1200 with the parallel algorithm) Results are presented in Section 5.1.

4.3 Experiment II: Evaluation of the Use of Constraints in the PSP

4.3.1 Motivation and Objective. The literature search indicates a large body of knowledge is being assembled about the PFP and PSP (Section 2.3). Intuitively, it seems this knowledge can be exploited to enhance the effectiveness, and efficiency of the search process. Validation of this conjecture is this experiment set's objective.

4.3.2 Methodology. The first step was the development of a "reasonable" set of domain constraints, which is detailed in Section 3.1.2. Two molecules, [Met]-enkephalin and Polyalanine were used in this experiment. The constraint sets: *none*, *loose*,and *tight*, are used for each molecule. Of course, the constraints *[Met]-enkephalin tight* are different than *Polyalanine tight*.

4.3.3 Parameter Selection. A group (or block) of five experiments are performed for molecule using each constraint set . In each block, all variables are consistent except the random number generator seeds. The seed used are in Table 6. Full parameter details are presented in Tables 8 and 9. Discussion of selection rational follows.

Based on insight from experimental results in Section 5.1, *reference population* sizes of 20 and 50 are used. Because the ratio of the feasible solution space, $\mathcal{F}$, to the search space, $\mathcal{S}$, GENOCOP-III cannot

Table 8. Input Parameters for Experiment II, Met-enkephalin

Parameter	none	loose	tight
Number_Variables	24	24	24
Number_NLE	0	0	0
Number_NLIE	0	18	18
Number_LC	0	0	0
Number_DC	24	24	24
Ref_Pop_Size	20	20	20
Search_Pop_Size	20	100	40
Number_Operators	10	10	10
Total_Evaluations	50,000	50,000	50,000
Reference_Period	10	35	30
Reference_Offspring	5	17	10
Select_Reference_Pt	0	1	1
Repair_Method	0	1	1
Replace_Prob	0.25	0.5	0.5
Reference_Init_Type	1	1	1
Search_Init_Type	1	1	1
Object_Type	1	1	1
Test_Case	10	14	15
Epsilon	0.0001	0.0001	0.001
Frequency_Mode	0	0	1

randomly generate an initial reference population in the required number of tries,[1] therefore inital populations are supplied. A total of eight population where generated. There is a population of 20 randomly generated unique points for satisfying the *loose* or *tight* constraints for each molecule. The population of size 50 contains duplicates of 25 random generated unique points satisfying the respective constraints. The permutations of the seeds in Table 6 were used in the random generation. Results are presented in Section 5.2.

4.4 Experiment III: Evaluation of Exogenous Parameters in the REGAL System

4.4.1 Motivation and Objective. The impact of exogenous control parameters on the effectiveness of GENOCOP-III has not been extensively studied. Exogenous parameters are those external to the algorithm. Consider, population size and number of domain constraints. Clearly, if we change the number of domain constraints, we are solving a different problem. On the other hand, if we change the population size, the behavior will change, likely resulting in a different answer, but to the same problem.

The experiments in Section 4.3 treated these parameters in an ad-hoc fashion, however, they were consistent for each group of experiments. Only the seed values were changed within a group. The results

[1] Based on the ratio of $\frac{\mathcal{F}}{\mathcal{S}}$, it would require 10^{67} tries to randomly generate just on fully feasible chromosome when using the *tight* constraint set for Polyalanine.

Table 9. Input Parameters for Experiment II, Polyalanine

Parameter	none	loose	loose	tight	tight	tight (relaxed terminals)
Number_Variables	56	56	56	56	56	56
Number_NLE	0	0	0	0	0	0
Number_NLIE	0	56	56	56	56	48
Number_LC	0	0	0	0	0	0
Number_DC	56	56	56	56	56	56
Ref_Pop_Size	20	20	50	20	20	20
Search_Pop_Size	40	80	100	40	50	40
Number_Operators	10	10	10	10	10	10
Total_Evaluations	50,000	20,000	50,000	20,000	50,000	50,000
Reference_Period	50	40	115	24	75	30
Reference_Offspring	5	18	30	10	25	10
Select_Reference_Pt	1	0	0	1	1	1
Repair_Method	1	0	0	1	1	1
Replace_Prob	0.25	0.50	0.50	0.50	0.50	0.05
Reference_Init_Type	1	1	1	1	1	1
Search_Init_Type	1	1	1	1	1	1
Object_Type	1	1	1	1	1	1
Test_Case	20	24	24	25	25	26
Epsilon	0.001	0.01	0.01	0.0001	0.0001	0.0001
Frequency_Mode	0	1	1	1	1	1

between groups, it seems, were effected by the variations in these parameters. There is extremely little published data on this issue. The success of REGAL for PSP is dependent on managing these parameters' effects. Thus, the objective of this experiment set is the characterization of effect of the exogenous parameters on the REGAL/GENOCOP-III system, at least with respect to the PSP.

4.4.2 Methodology. Michalewicz has publish results using earlier variants of the algorithm (85). Frequently reported parameters are 70 for population size, .20 for Probability of replacement, and 28 for number of offspring generated per "generation". Most experiments ran for 5000 *iterations*. Given that GENOCOP is a *steady state* GA, it was not clear whether this term meant evaluations or generations. Personal communications with the co-creator of GENOCOP-III, Girish Nazhiyath verified that iterations is approximately equivalent to generations. Therefore, 5000 iterations equals approximately 350,000 evaluations with a population size of 70. Previous experiments had produced good results with 20-40,000 evaluations. Therefore, 100,000 evaluations were used in this experiment set.

It is highly likely the static parameters and the operator probability distribution are confounded. In attempt to decouple them, the static parameters are tested while the operator frequency control is set to *adaptive*. This allows the frequency of the operator to "float" in response to the needs of the system. After the other parameters have been analyzed, specific operator frequencies can be studied.

The experiment set consist of a full factorial design using treatment levels shown in Table 10. Some combinations are not practical, specifically, generating more than the reference population size number of offspring during a reference population evaluation. The result is total of 540 experiments!

4.4.3 Exogenous Parameter Evaluation Experiments. A total of 540 experimental cells are defined from permutations of the parameter values in Table 10. This is a full factorial design with the except that permutation with the number of *Offsprings* greater than the *Reference Population Size* where eliminated. This combination is not practical because it would force evolution of only the reference population.

Table 10. Exogenous Parameter Evaluation Experimental Values

Parameter (Short Name)			
Reference Population Size (Ref Pop)	20	50	
Search Population Size (Search Pop)	20	50	70
Periodicity of Reference Population Evaluation (Periodicity)	50	100	150
Offspring per Reference Population Evaluation (Offsprings)	10	20	30
Probability of Replacement for Search Population (Probability)	0.05	0.20	0.50
Selection of Reference Point for Repair (Ref Point)	0=random	1=ordered	
Repair Method (Repair)	0=random	1=deterministic	

The results are subjected to the twelve hypothesis test detailed in Table 11. Results are analyzed in Section 5.3.

Table 11. Exogenous Parameter Evaluation Hypothesis Tests

Test Number	H_0, Not Significant H_1,Significant
1	Reference Population Size
2	Search Population Size
3	Periodicity of Reference Population Evaluation
4	Offspring per Reference Population Evaluation
5	Probability of Replacement
6	Selection of Reference Point for Repair
7	Repair Method
8	Reference Population Size x Search Population Size
9	Reference Population Size x Repair Method
10	Reference Point Selection vs Repair Method
11	Offspring per Reference Population Evaluation x Replacement Population
12	Probability of Replacement x Repair Method

4.5 Experiment IV: Evaluation of Para-REGAL

4.5.1 Motivation and Objective. Results for the REGAL experiments were far better than expected. An obvious extension of of serial REGAL is Para-REGAL. However, the behavior of a parallel GA is frequently different than the serial version. Additionally, little is know about the behavior of GENOCOP-III in

a parallel implementation. Thus the objective of this experiment set is an initial appraisial of Para-REGAL with respect to the new migration parameters.

4.5.2 Methodology. In terms of input parameters, etc., Para-REGAL inherits the charateristics of REGAL. Additionally parameters are required for the number of islands, probability of migration and probability of complete migration. For this experiment set [Met]-enkephalin is used as the test molecule becuase of its rugged fitness surface. Based on insight from Experiments I–III, an input paramter file is defied that contains varing values for the different nodes. For example, *Island 0* will have no constraints, *Islands 1 & 2* will use *loose* constraints, and *Island 3* will use the *tight* constriants. to insure randomness, seeds are taken from a table of random number in Montmogery (87). Also, are results better if the same total number of eveluations is distributed over multiple processors? To answer this question, the number of evaluations used in Experiment III, $100,000$, will be divided by the number of islands.

4.5.3 Para-REGAL Experiments. The initial experiment suite is for four islands. The set $\{0.0, 0.33, 0.66, 1.00\}$ is used for P_m and P_{cm}. Each island will evolve for $25,000$ evaluations. An initial suite of 16 experiments are defined (Table 12). Results are presented in Section 5.4.

Table 12. Para-REGAL Experiment for Four Islands

Parameter	Island 0	Island 1	Island 2	Island 3
Num Nonlinear Constraints	0	5	18	18
Ref & Search Pop Size	50	50	50	50
Periodicity	250	250	250	250
Test Case Number	10	11	14	15
Constraint Set	None	ω's Only	Loose	Tight
Num Offspring	20	20	20	20
Replacement Ratio	0.5	0.5	0.5	0.5
Seed 1	10480	15011	01536	02011
Seed 2	14577	25417	09922	20655

Exogenous parameters applicable to all runs

4.6 Summary

After discussion of generic experiment design issues, this chapter has detailed four separate experiment sets. Experiment I is designed to characterize the performance of the farming model parallel hybrid GA versus the serial hybrid GA. Experiment II is designed to characterize the use of domain knowledge in the form of constraints in REGAL, a real-valued GA. Experiment III is designed to characterize and identify good values for the exogenous parameters used in REGAL. Finally, Experiment IV is designed to evaluate

migration parameters and compare the performance of Para-REGAL versus the best results found using REGAL.

V. Results and Analysis

This chapter contains the summarized results and analyzes the principle experiments conducted as part of this research. Raw data is not included, but is available in the Auxiliary Volume (59)

5.1 Experiment I: Parallel Hybrid GA

The results from this experiment set is first analyzed with respect to effectiveness, Section 5.1.1. The next analysis considers efficiency, first using the serial executions as baseline, Section 5.1.2, and then comparing with the parallel results, Section 5.1.3. Portions of this experiment set were presented at the 1996 National Meeting of the American Chemical Society (60).

5.1.1 Effectiveness Analysis. The first use of the results from this experiment set is a comparison of the hybrid GA sub-types. Table 13 presents the final fitness values after 2000 evaluations. The trajectories, defined by the current best solution, are consistent! For example, best fitness for a given set of parameters is the same at the say, 20th generation, regardless of whether the experiment is running to 1000 or 2000 evaluations. Likewise, both serial and parallel executions generate the same fitness trajectory, generation by generation. This satisfies the correctness requirement.

Table 13. Final Fitness Values for Experiment I after 2000 Evaluations

Algorithm Sub Type	Population Size 20	Population 50 Size	Population Size 100
FPBald	-21.23	-2.62	9.50
FPLam	-30.84	-22.26	-15.99
TSLam	-28.73	-28.16	-27.68
FPSGA	-13.51	23.5487	35.21
TSSGA	3.43	-19.45	-1.43

For both the trajectory and runtime plots, the following labels apply: FPBald for Hybrid GA w/Fitness Proportional Selection and Baldwinian Minimization, FPLam for Hybrid GA w/Fitness Proportional Selection and Lamarckian Minimization, TSLam for Hybrid GA w/Tournament Selection and Lamarckian Minimization, TSSGA for SGA w/Tournament Selection, and FPSGA for SGA w/Fitness Proportional Selection. Consistent with other results (35), the population size of 20 produces the best results when using Fitness Proportional (FP) selection, with or without minimization. One explanation suggests the smaller size increases *selection pressure* on the population. Another explanation suggest since the execution runs for x evaluations, $\frac{x}{pop\_size_{small}} > \frac{x}{pop\_size_{large}}$. Thus with the smaller population size, it suffers more generations, and, therefore, applications of operators. The population size of 50, on the other hand, produces the best results when using Tournament Selection (TS). In TS, individuals complete against each other, rather than the population as a whole inducing even stronger selection pressure than FP. A larger population provides a

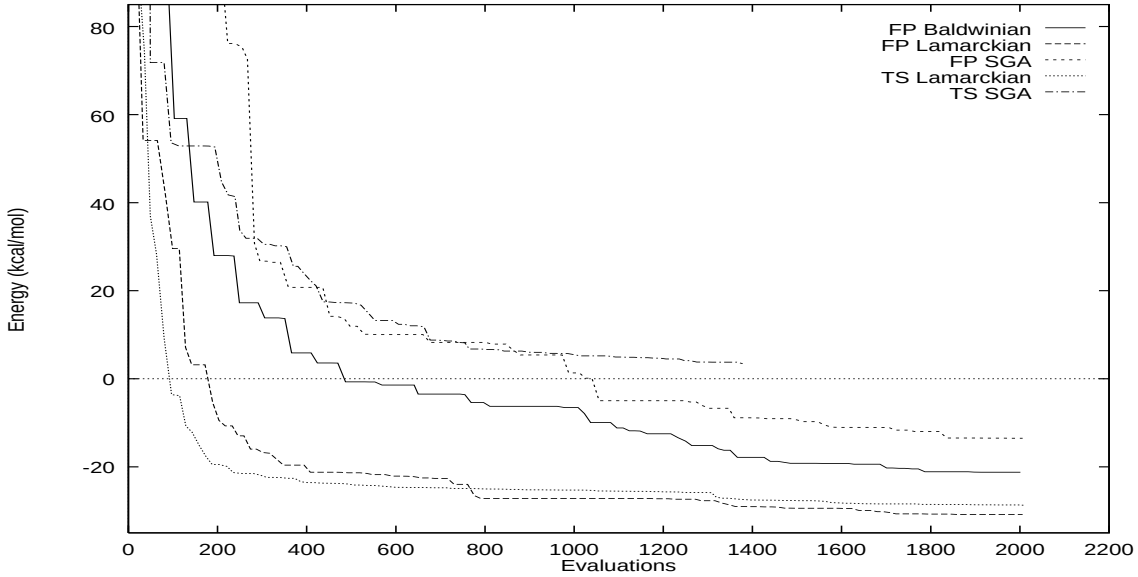


Figure 10. Trajectory Plot, Population Size of 20

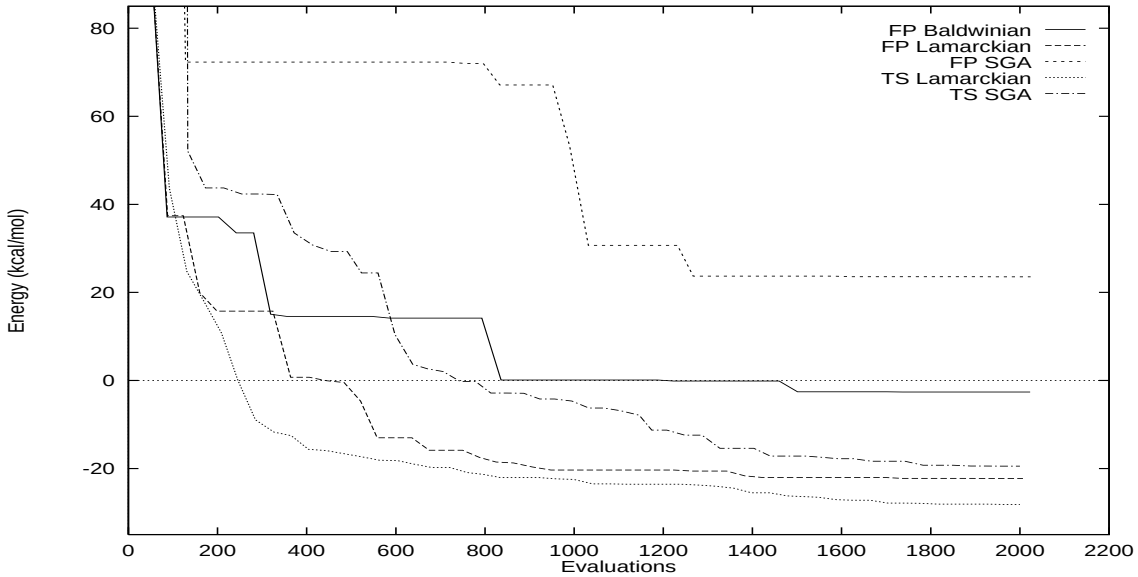


Figure 11. Trajectory Plot, Population Size of 50

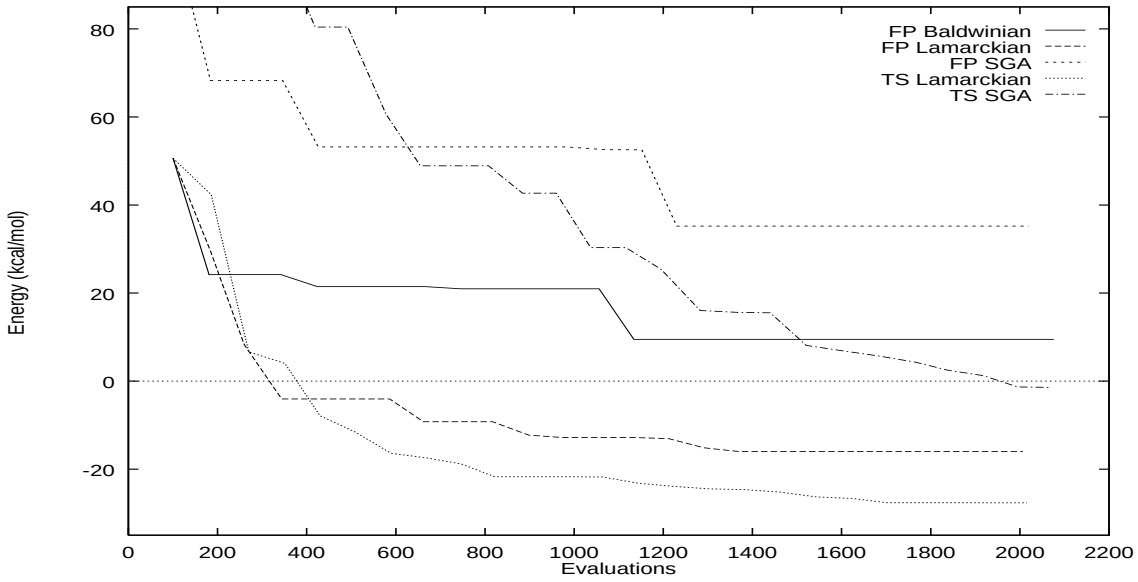


Figure 12. Trajectory Plot, Population Size of 100

more diverse selection of genetic material from which to select. However, TS usually converges prematurely when used for PSP. Notice the trajectory for TSSGA terminates after 1300 evaluations in Figure 10. To summarize, fitness proportional selection, population size 20, and Lamarckian local minimization have been the most effective hybrid GA examined. Therefore, it will be used as the benchmark against which the effectiveness of other refined GAs for PSP will be measured.

5.1.2 Efficiency Analysis, Serial. The average run times of the various GA subtypes are presented, first grouped by population size, Figures 13–15, then grouped by subtypes, Figures 16–20. Note that Figures 13–15 actually contain five lines each. The FPSGA and TSSGA lines are buried at the bottom of each plot. This emphasizes the difference in execution time between the minimized and non-minimized GAs. Regardless, because of greater effectiveness, minimized GAs are the focus of this effort. At all three population sizes, the run time increases linearly with the slope increasing as the population size decreases. This is explained by the smaller population size experiencing more generations and likewise more total overhead from each generation. When viewed across the population sizes by GA type, Figures 16–20, the run time increase is also linear. The smaller population size continues to generally require a longer run time. The slopes of the minimized GAs are steeper and more varied than the non-minimized GAs. The flattening of TSSGA population size 20, Figure 19 is explained by the execution terminating prematurely after approximately 1300 evaluations because of convergence.

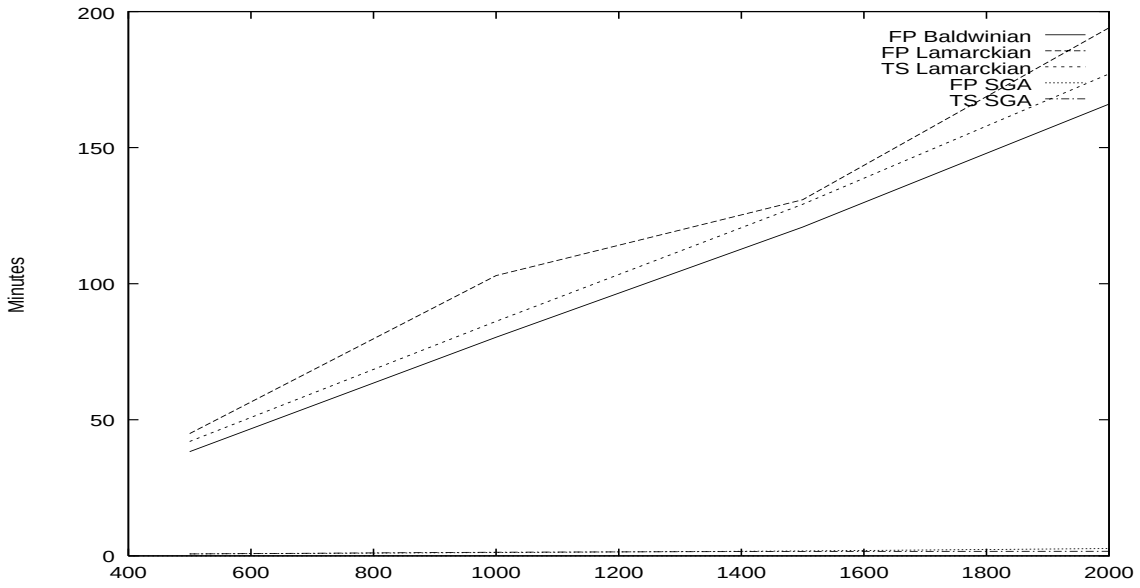


Figure 13. Serial Run Time, Population Size = 20

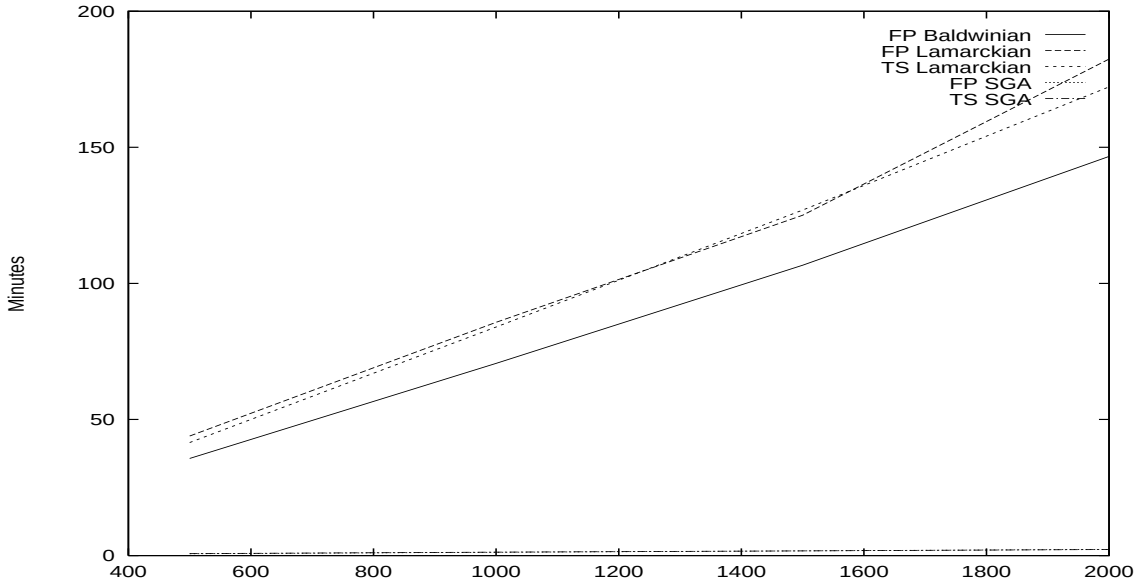


Figure 14. Serial Run Time, Population Size = 50

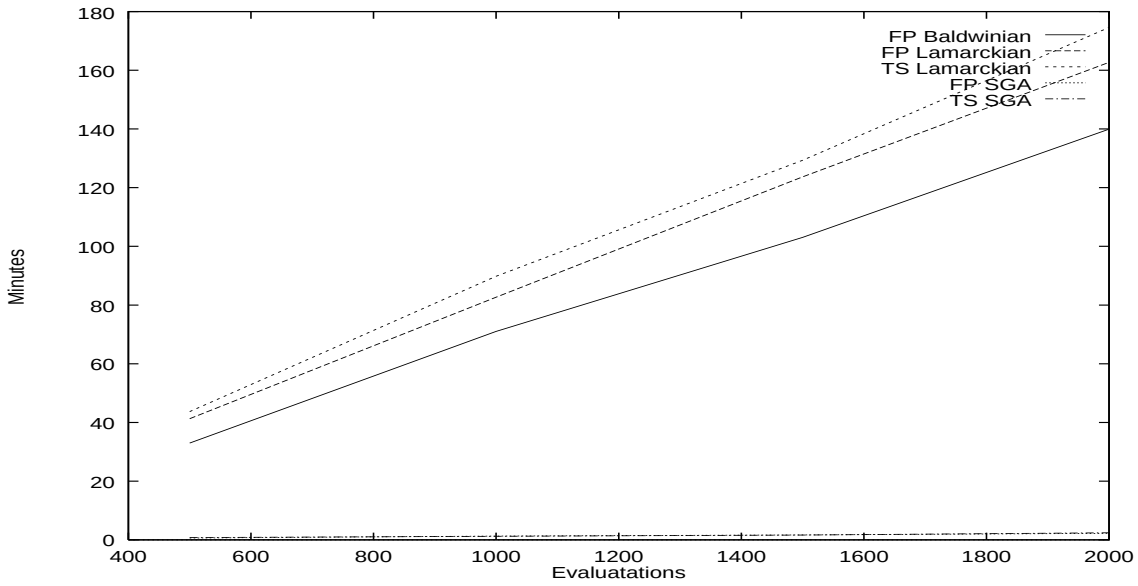


Figure 15. Serial Run Time, Population Size = 100

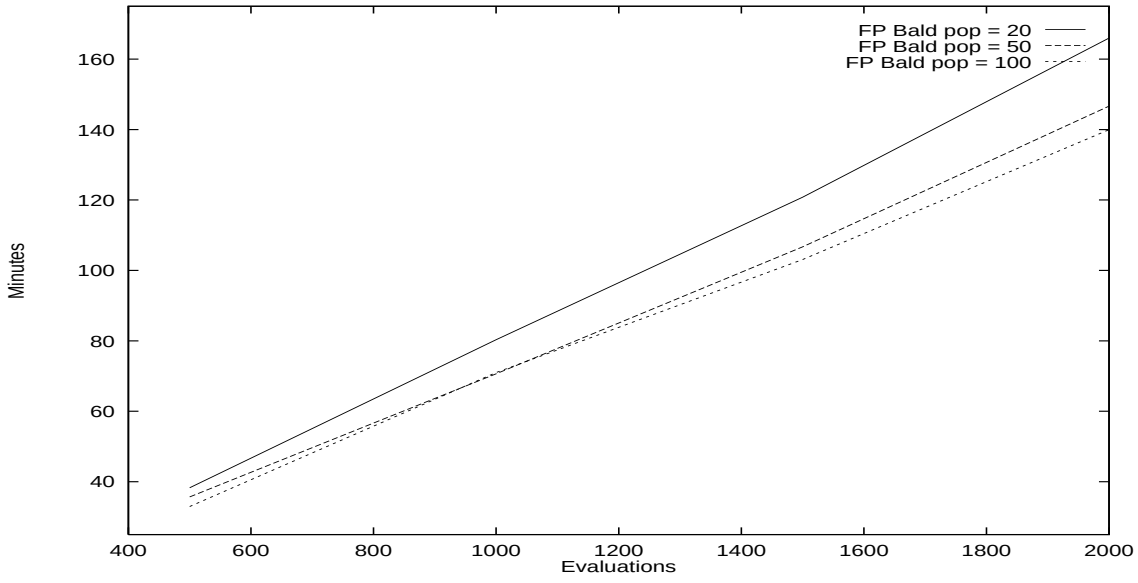


Figure 16. Serial Run Time, Fitness Proportional Baldwinian, by Population Size

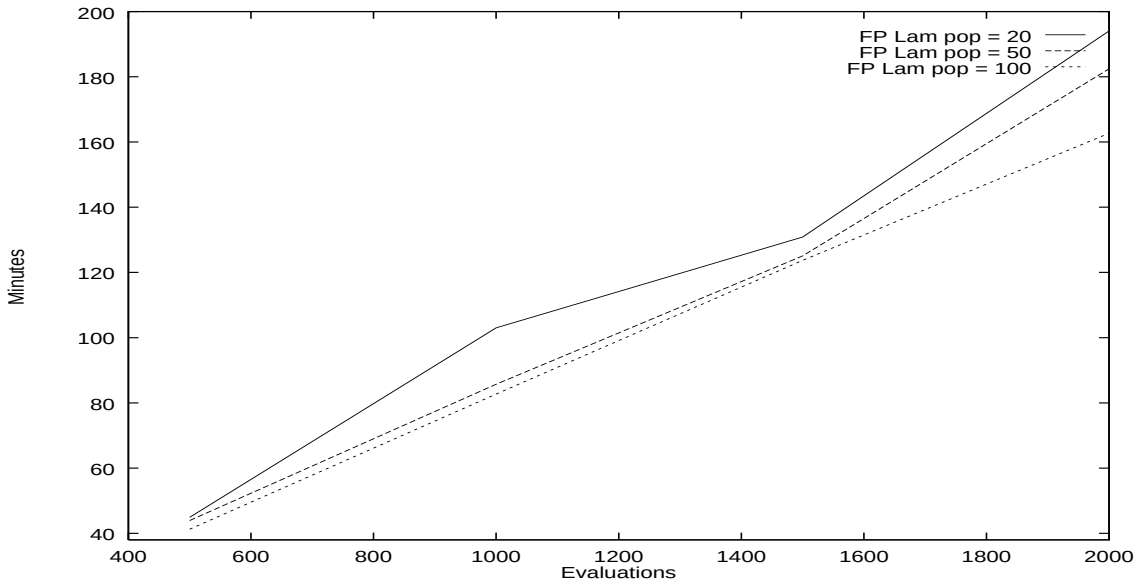


Figure 17. Serial Run Time, Fitness Proportional Lamarckian, by Population Size

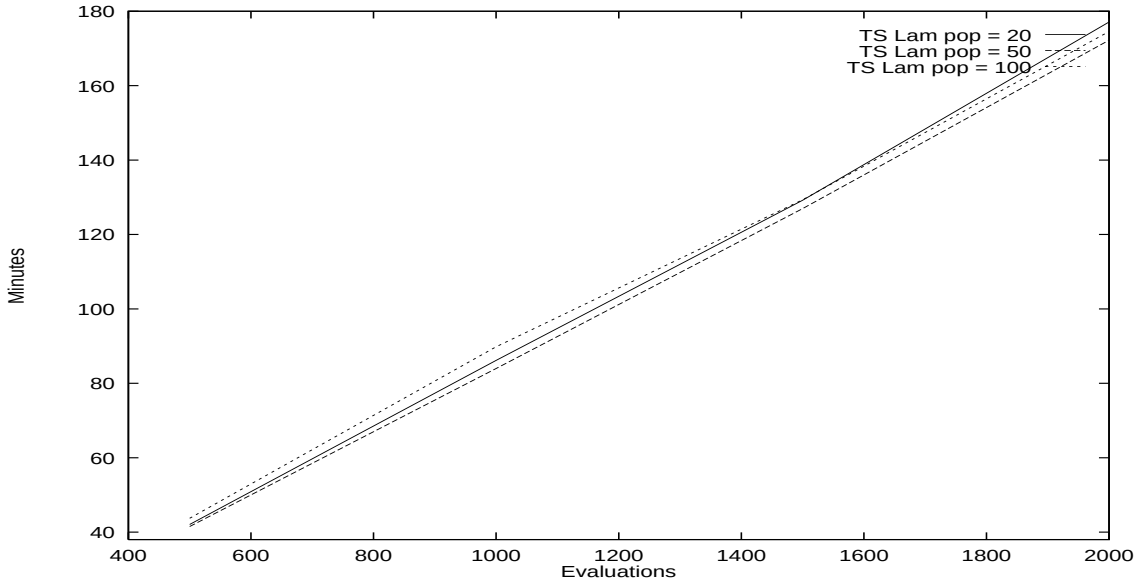


Figure 18. Serial Run Time, Tournament Selection Lamarckian, by Population Size

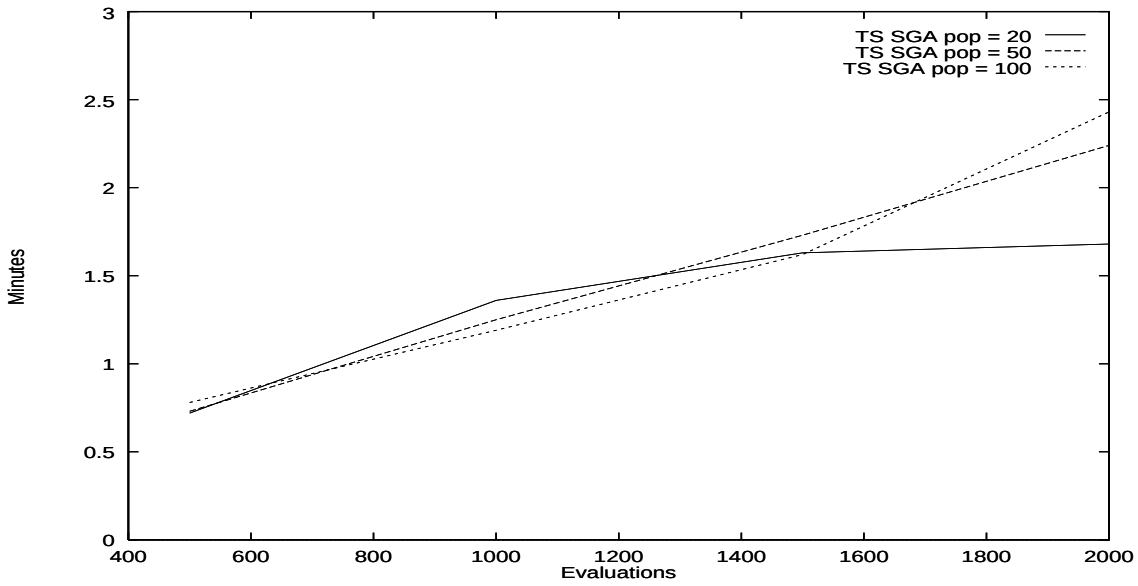


Figure 19. Serial Run Time, Tournament Selection SGA, by Population Size

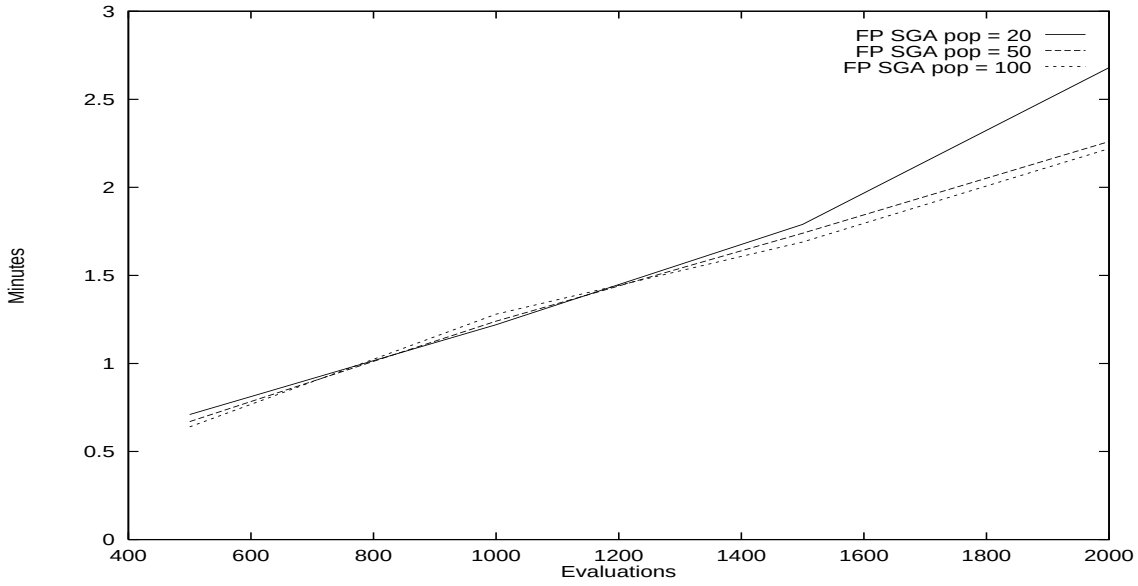


Figure 20. Serial Run Time, Fitness Proportional SGA, by Population Size

5.1.3 Efficiency Analysis, Parallel. In this section cost/benefit aspects of the parallel implementation are examined. First, an empirical look at the additional overhead cost induced by communication. Recall that the work performed by the serial implementation is divided in the parallel implementation cleanly between the client and server nodes. Thus, a client with just one server should execute no faster than the serial version. In fact, the client-server should actually take longer because of communication cost between the client and single server.

$$C_{communications} = T_{P_2} - T_S \tag{11}$$

where $C_{communications}$ is the cost, in time, of the communications and T_{P_2} is the average runtime of the parallel implementation with 2 nodes (a client and one server) and T_S is the average runtime of the serial implementation. As a unit-less measure of the communications cost, I define the Communication Cost Index to be the ratio of the Client-Server pair runtime to the Serial implementation.

$$Communication\ Cost\ Index = \frac{T_{P_2}}{T_S} \tag{12}$$

Figures 21–23 plot the communication cost index values by population size.

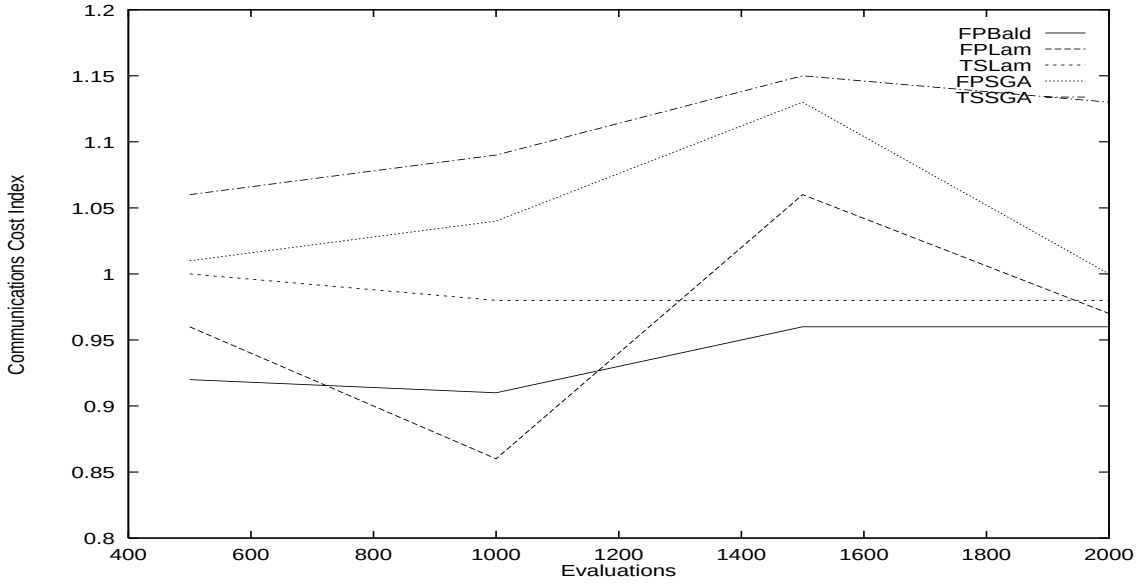


Figure 21. Communications Cost Index, Population Size 20

Speedup is a measure capturing the relative benefit of solving a problem in parallel. It is defined as

$$S = \frac{T_S}{T_P} \tag{13}$$

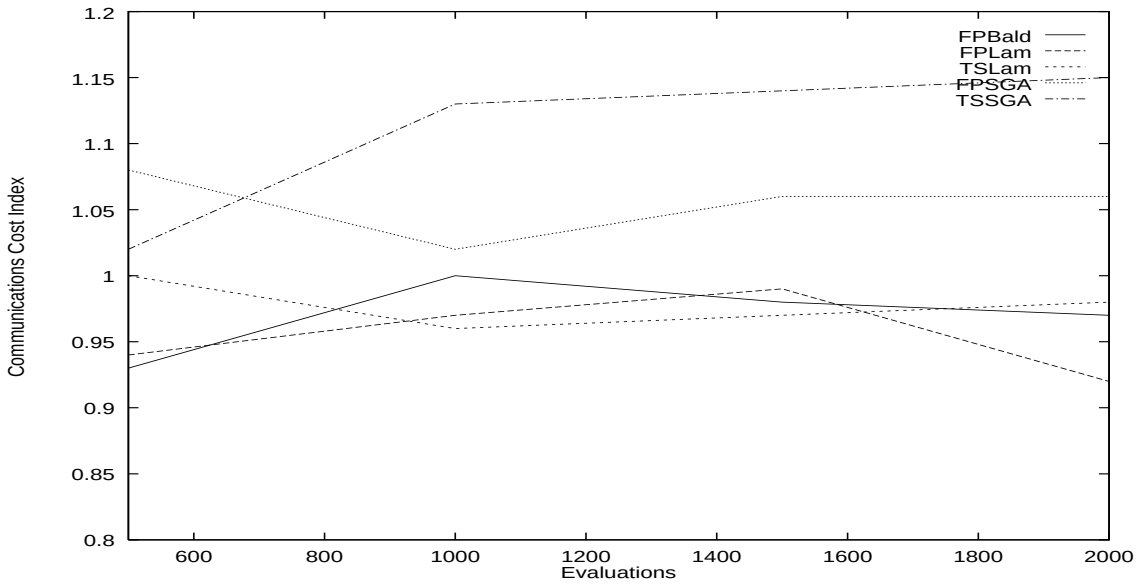


Figure 22. Communications Cost Index, Population Size 50

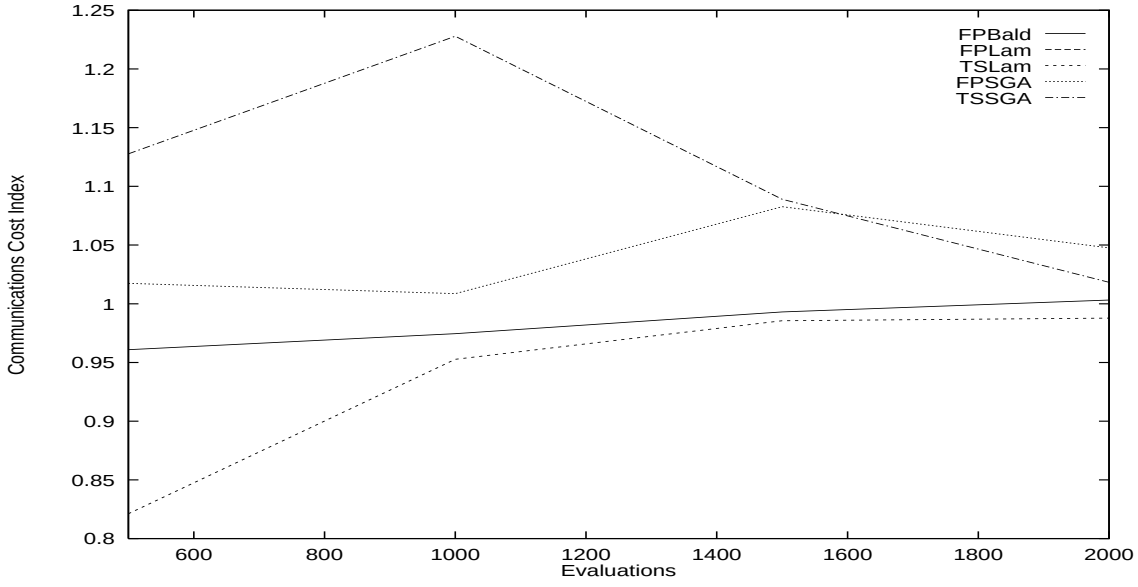


Figure 23. Communications Cost Index, Population Size 100

where S is the speedup, T_S is the time of the *best* serial algorithm for solving the problem, and T_P is time taken by the parallel algorithm (66). In *linear speedup* S increases proportionally with the number of processors p. If $S > p$, it is called *super linear speedup*. While sometimes observed, it indicates T_S is from a non-optimal serial algorithm. Speedup for PHGA is plotted by population size in Figures 24–26. Efficiency

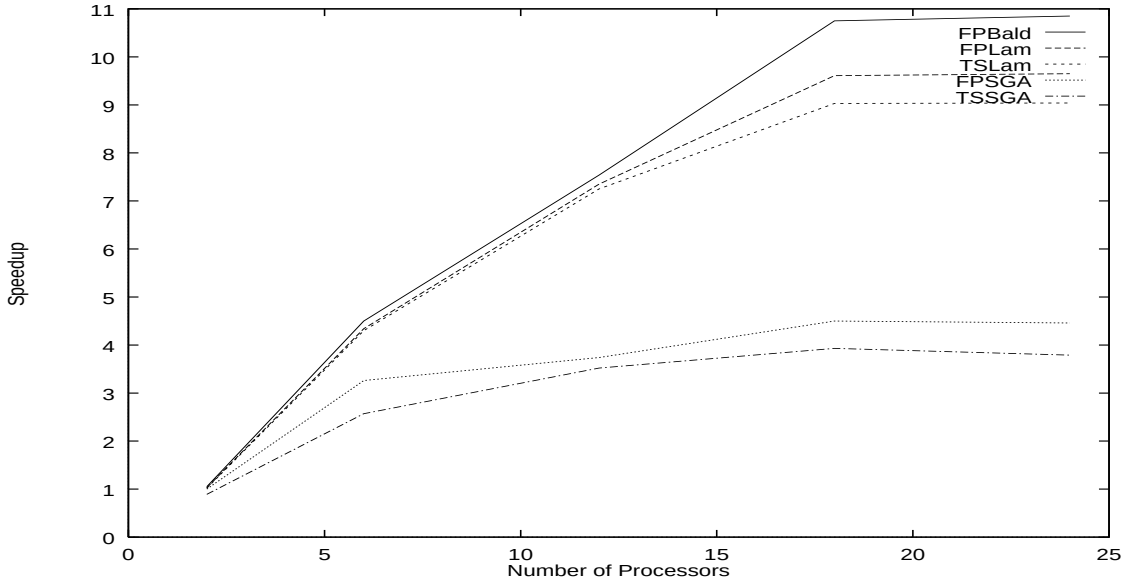


Figure 24. Speedup, Population Size 20

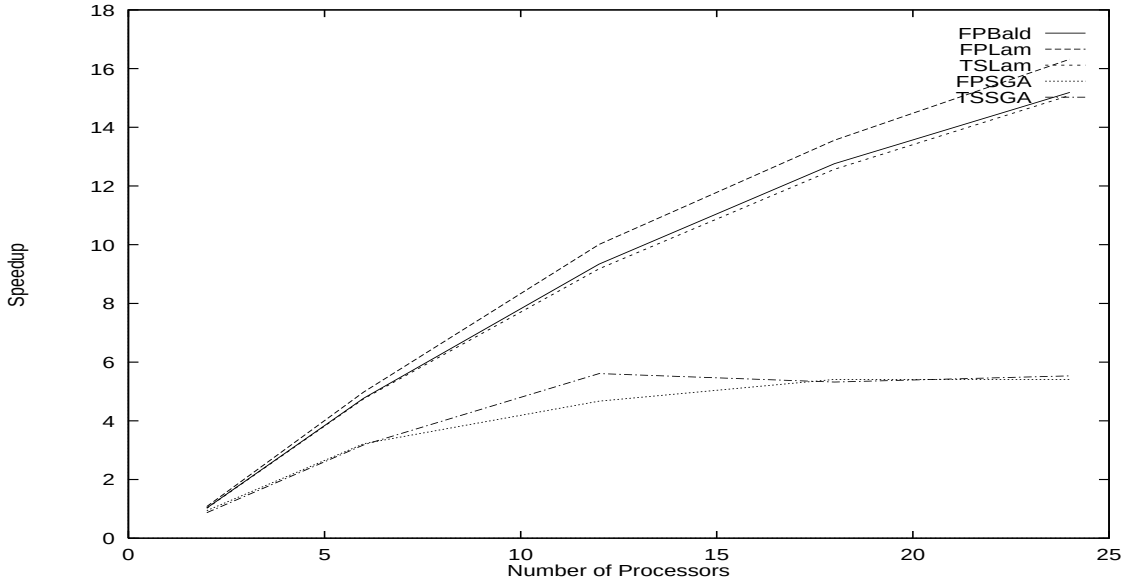


Figure 25. Speedup, Population Size 50

is a measure of the fraction of the fraction of time for which each processor is usefully employed. It is defined

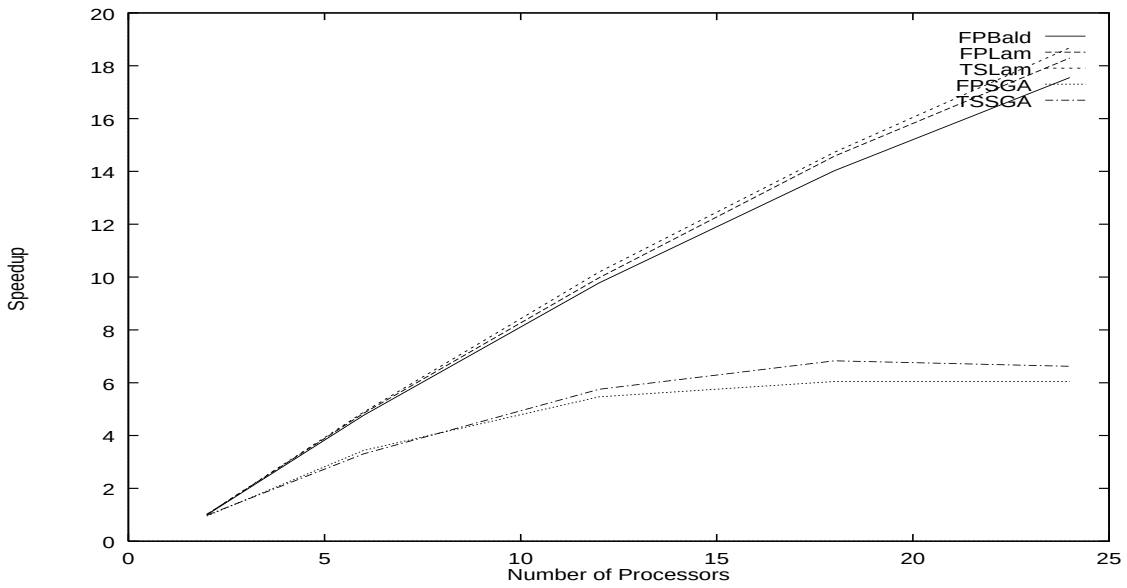


Figure 26. Speedup, Population Size 100

as

$$E = \frac{S}{p} \tag{14}$$

where E is the efficiency, S is the observed speedup, and p is the number of processors. Efficiency for PHGA is plotted by population size in Figures 27–29. *Cost* is the product of the parallel run time and the number of processors. A parallel system is said to be *cost-optimal* if the cost of solving a problem is proportional to the execution time of the fastest-known sequential algorithm on a single processor. Since efficiency is the ratio of sequential cost to parallel cost, a cost-optimal parallel system has an efficiency of $\Theta(1)$. PHGA is reasonably efficient when the number of processors is roughly half the population size. While, it is not cost-optimal, it will perform identically to the serial version only faster. Most parallel GAs do not have this feature of identical behavior. As a result, previously (and expensively) gained insight into exogenous parameter selection does not transfer to the parallel implementation. This is not a characteristic of PHGA.

5.2 Experiment II: Preliminary REGAL Evaluation

This section presents the results of experiments on two separate molecular models, [Met]-enkephalin and a 14 residue model of Polyalanine. Portions of this experiment set were presented at the Midwest Regional Meeting of the American Chemical Society (61, 62), and have been accepted for the proceedings of ACM's 1997 Symposium on Applied Computing (SAC'97) (63). In previously published results (78), fitness proportional (FP) selection with binary encoding was shown most effective for this particular problem, at

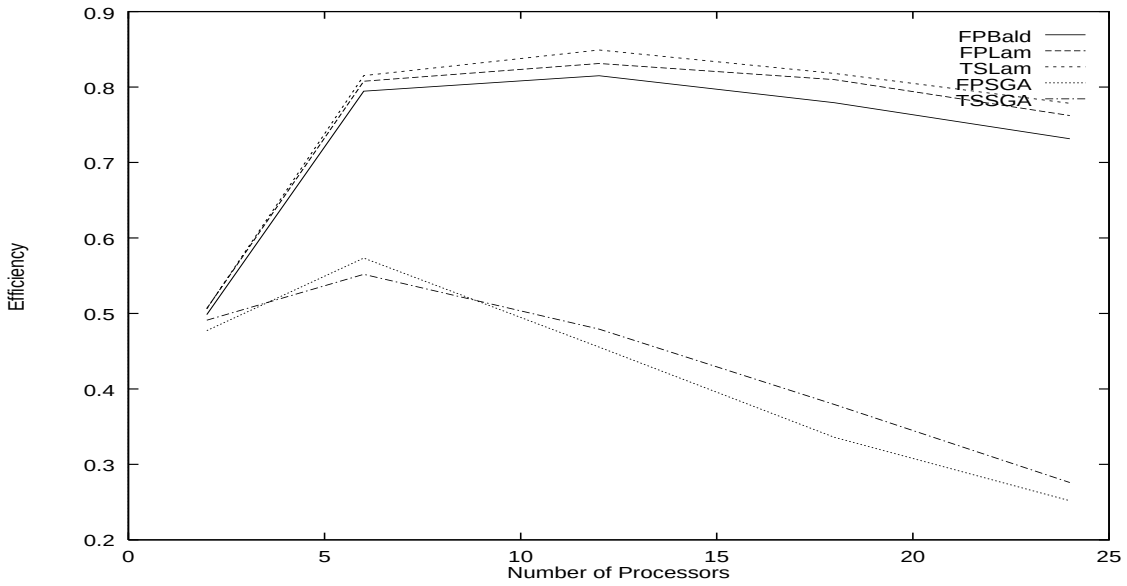


Figure 27. Efficiency, Population Size 20

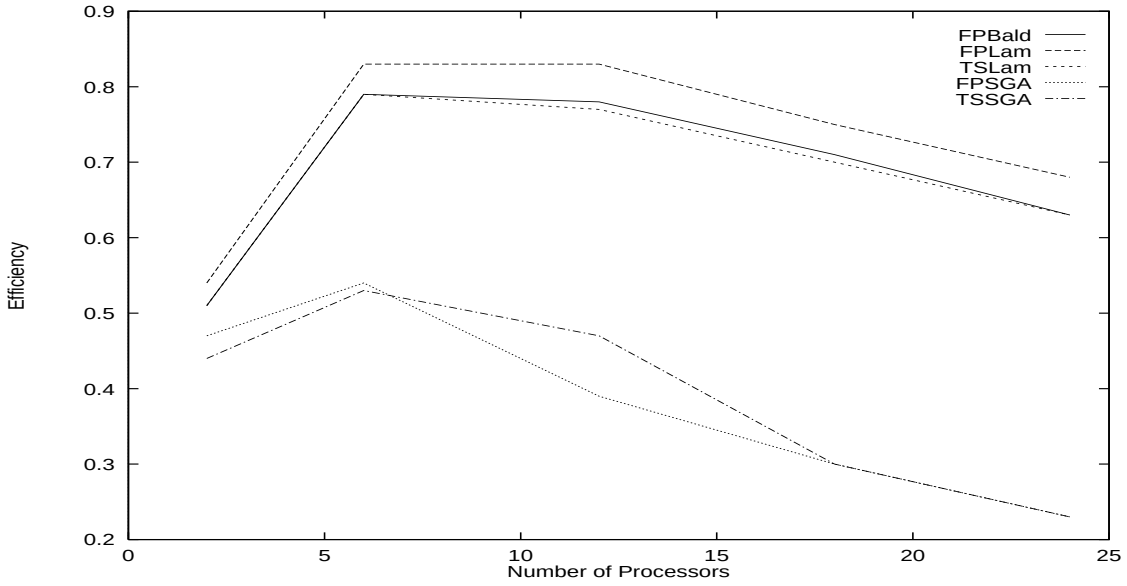


Figure 28. Efficiency, Population Size 50

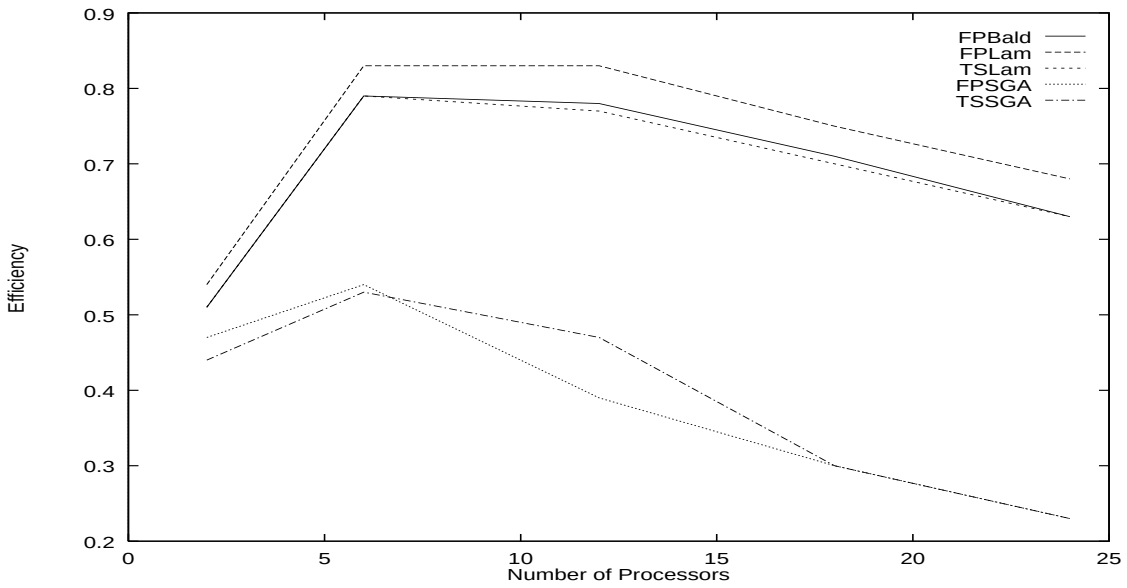


Figure 29. Efficiency, Population Size 100

least with respect to [Met]-enkephalin. This selection technique for binary encoding is compared with the REGAL approach.

5.2.1 [Met]-enkephalin. In general the hybrid GA has been more effective than the REGAL approach in minimizing [Met]-enkephalin with a best average of -28.35 kcal/mol (Table 14) verses -26.38 kcal/mol (Table 15). However, the best value in this experiment set, -30.32 kcal/mol, was a REGAL experiment, using no constraints and Lamarckian minimization. This single example demonstrates a potential for local minimization incorporated with REGAL. But in general, tighter constrains appear to interfere with local minimization. That is, a local minima is found during the initial evaluation from which the experiment is unable to escape. I suspect that as the ratio of feasible space $\mathcal{F}$ to search space $\mathcal{S}$ gets smaller, the operators are unable to generate a more fit "feasible" candidate, and thus, escape the local minima.

Table 14. Final minimum energies (kcal/mol) for [Met]-enkephalin, using binary GA

Algorithm	Mean	Std. Dev.	RMSD Best
SGA	-22.58	1.57	4.51
Baldwinian	-22.57	1.62	3.96
Lamarckian	-28.35	1.29	3.33

It is interesting to note that conformers have been identified with values less than the accepted optimal conformation (CHARMM equivalent of the ECEPP/2 conformation of Li and Scheraga (74)). It is believed the optimal conformation for ECEPP/2 and CHARMM are different.

Table 15. Final minimum energies (kcal/mol) for [Met]-enkephalin, using REGAL

Algorithm	Mean	Std. Dev.	RMSD Best
No constraints	-24.92	2.99	4.57
No constraints w/local min	-26.38	2.69	4.40
Loose constraints	-22.01	2.69	4.25
Loose constraints w/local min	-24.95	4.23	4.26
Tight constraints	-23.55	1.69	3.23
Tight constraints w/local min	-17.71	0.50	5.05

5.2.2 Polyalanine. The effectiveness of the Binary GA (even with minimization) did not hold for the larger molecule Polyalanine (Table 16). Significant improvement was observed when the *step* size in the conjugate gradient minimization was properly sized for for the larger molecule (another example of using "domain knowledge"). When examined visually, these conformations did not appear to be forming the expected α-helix secondary structures. With adequate domain knowledge, in the form of tight constraints, REGAL performs well on the larger molecule (Table 17). When allowed to reach 150,000 evaluations, the energy value is almost that of the optimal conformation without relaxation of bond lengths and bond angles. When examined visually, these conformations definitely formed the expected α-helix secondary structures. Again, local minimization was not effective when used in conjunction with constraints. This time, the difference between the results is more substantial.

Table 16. Final minimum energies (kcal/mol) for Polyalanine, using binary GA

Algorithm	Mean	Std. Dev.	RMSD Best
SGA	-93.25	10.85	9.67
Baldwinian	-103.73	16.5	7.36
Lamarckian	-140.60	5.39	12.74
Lamarckian corrected	-308.51	8.26	5.03

5.2.3 Efficiency. The rns conducted in this experiment set were conducted on a variety of platforms. They include a 368 node Paragon supercomputer, 100 and 200 mhz Silicon Graphics workstation, SUN Sparc workstations (2, 5, and 20), and SUN Ultra Sparc workstations.The bulk of the effort was accomplished in a common user lab of 46 networked Sparc20 workstations. As is expected, run times (wall clock) varied with system loading. However, a few general observations can be made.

- Met-enkephalin
  - Lamarckian minimized Binary GA, average run time = 13 hours

Table 17. Final minimum energies (kcal/mol) for Polyalanine, using REGAL

Algorithm	Mean	Std. Dev.	RMSD Best
No constraints	-273.08	13.81	6.25
Loose constraints	-336.65	4.50	1.87
Loose constraints w/local min	-309.00	8.19	2.70
Tight constraints	-337.64	4.40	0.98
Tight constraints w/local min	-316.47	0.0[1]	1.17
Tight constraints w/ relaxed terminals	-338.30	4.24	1.42
Tight, relaxed 150K evals	-351.76	0.57	1.40

  - REGAL, average run time = 2 hours

- Polyalanine
  - Lamarckian minimized Binary GA, average runtime = 120 hours
  - REGAL, average run time = 4 hours

While results prove nothing, initial data suggest the REGAL approach scales better than the binary GA with local minimization. While the above times might seem excessive, it takes years to identify protein conformations using experimental methods such as crystallography.

5.3 Experiment III: Analysis of Exogenous Parameters for REGAL

This experiment set analyzes the effect of the *independent* static input parameters identified in Section 4.4.3 on behavior of the REGAL system for PSP. These parameters are called independent because they are not tied to a specific problem instance unlike, for example, the number of non-linear inequalities. The parenthetical comments in the following paragraphs indicate the "handle" that can be used to link the parameter being discussed to the data tables. Summarized data is presented in Table 18. ANOVA[2] results are presented in Table 19. The parameters for the experiments yielding the 5 lowest energies are shown in Table 20. The plot of the *current best trajectory* of these 5 experiments are shown in Figure 30.

Analysis 1 considers the reference population size (Ref Pop) parameter. On the average, lower energies are found in experiments with a reference population size of 50. The average runtime is greater with the reference population size of 50, which was expected due to the sorting time of the larger population.[3] The

[2] Analysis of Variance, see Appendix F.1 for more details. Concern has be raised about lack of variability because a single seed set was used. The Kruskal-Wallis H Test (Appendix F.2) was used as an independent method to verify the ANOVA results. The conclusions were the same. Kruskal-Wallis results are not shown

[3] Hypothesis testing was not done on run times because system loading in the multi-user environment could not be controlled. They are provided for reference only. However, the large number of experiments tends to dampen out cases were the platform was heavily loaded. Thus, the data are insightful.

difference in population size was significant at the 95% confidence level but not at high confidence levels. ANOVA results are presented in Table 19.

Analysis 2 considers the search population size (Search Pop). In this test suit, the average fitness improves as the search population increases, however, the difference in population size is not significant at the 95% confidence level. Note the trend of average run times increase with the population size shown in the previous paragraph does not hold.

Analysis 3 considers the periodicity (Periodicity) of the reference population evaluation. That is, the reference population is evaluated[4] every time this set number of evaluations is performed, regardless of which population has been evaluated. Within the parameters of this experiment set, this input parameter is not significant.

Table 18. REGAL Input Parameter Analysis, Summary Data

Analysis #	Parameter	Value	Avg Fitness	Std Dev	Avg Run Time	Std Dev
	Overall		-24.607	1.99	3.78	1.57
1	Ref Pop	20	-24.29	2.01	3.44	1.66
		50	-24.65	1.96	4.01	1.48
2	Search Pop	20	-24.46	1.94	3.76	1.74
		50	-24.49	1.97	3.86	1.53
		70	-24.57	2.06	3.72	1.45
3	Periodicity	50	-24.72	1.97	3.85	1.72
		100	-24.36	2.07	3.86	1.49
		150	-24.44	1.90	3.63	1.49
4	Offsprings	10	-24.43	1.90	3.85	1.68
		20	-24.44	2.03	3.56	1.45
		30	-24.79	2.06	4.10	1.55
5	Probability	0.05	-23.80	1.73	3.63	1.38
		0.2	-24.78	1.92	3.62	1.48
		0.5	-24.95	2.10	4.09	1.80
6	Ref Point	Random	-24.42	2.04	3.76	1.41
		Ordered	-24.60	1.93	3.80	1.72
7	Repair	Random	-25.21	1.87	3.92	1.68
		Deterministic	-23.81	1.84	3.64	1.46

Fitness is in kcal/mol
Runtime is in hours

Analysis 4 considers the number of offsprings (Offsprings) to be generated for the reference population every time the reference population is "evaluated". Remember that members of the reference population

[4] This nomenclature is from the GENOCOP-III documentation. It would be more accurate to stay the reference population is operated upon.

must be fully feasible, thus there may be no new chromosome added to the reference population. Like periodicity, this parameter by itself, is insignificant.

Analysis 5 considers the probability (Probability) that a repaired candidate solution will replace its parent in the search population . Of all factors considered, this was the second most significant. Results here contradict results published by Michalewicz using GENOCOP-III for nonlinear optimization problems(85). He reports best results with a 0.20 probability of replacement. Of the three values tested, $\{0.05, 0.2, 0.5\}$, these results show best results, on average, with 0.5. Granted, we are discussing two different problems, and further more, these results are only applicable to the specific experiments run. Additional study is needed to fully characterize this parameter.

Analysis 6 considers the method (Ref Point) used to select the point, or individual, in the reference population that is used to "repair" a infeasible candidate solution from the search population. The options are to either select an individual based on a randomly generated number $[0.0, 1.0)$ (Random), or use the probability distribution of the reference population (Ordered). This parameter by itself is not significant.

Table 19. Phase I Input Parameter Analysis, ANOVA

Source of Variation	Sum of Squares	DOF	Mean Square	F_0	$\alpha = 0.05$
Ref Pop	16.28	1	16.28	5.016 *	3.92
Search Pop	1.21	2	0.61	0.1896	3.07
Periodicity	13.09	2	6.545	2.0510	3.07
Offsprings	10.98	2	5.49	1.7204	3.07
Probability	137.72	2	68.86	21.5785 *****	3.07
Ref Point	4.41	1	4.41	1.3820	3.92
Repair	262.54	1	262.54	82.27172 *****	3.9
Ref Pop x Search Pop	1.78	2	0.89	0.2789	3.07
Ref Pop x Repair	15.01	1	15.01	4.7037 *	3.92
Ref Point x Repair	0.52	1	0.52	0.1630	3.92
Offsprings x Probability	6.04	4	1.51	0.4732	2.45
Probability x Repair	3.93	2	1.97	0.6158	3.07
Error	1653.01	518	3.19		
Total	2126.52	539			

* Above indicates the source of variation is significant at the $\alpha = 0.05$ level. ***** indicates it is still significant at the $\alpha = 0.025$ level. That is, we reject the H_0 hypothesis that the population means are equal.

Analysis 7 considers the method (Repair) used to repair the candidate solution from the search population. Repair is attempted by generating a linear combination of the candidate solution and the reference point with a value α. The repair method parameter controls how α is determined. The options are to either randomly generate values $[0.0, 1.0]$ for α (Random), or repeatedly generate a bisection, $\alpha = 2^{-i}$ $1 \leq i \leq 20$, until a fully feasible candidate is generated (Deterministic). This parameter is the most significant source of variance, with the random option producing the best results.

Analysis 8 considers the interaction between the size of the reference (Ref Pop) and search populations (Search Pop). Individually, the reference size is significant, but not the search size. As a source variation, the interaction between these two factors is insignificant.

Analysis 9 considers the interaction between the reference population size (Ref Pop) and the choice of repair method (Repair). Individually, both of these factors are significant, with the choice of repair method highly significant. Their interaction is also significant, but only at the $\alpha = 0.05$ level.

Analysis 10 considers the interaction between the method (Ref Point) used to select the point, or individual, in the reference population that is used to "repair" a infeasible candidate solution from the search population and the choice of repair method (Repair). Individually, the first factor was not significant while the second was extremely significant. Their interaction is an insignificant source of variation.

Analysis 11 considers the number of offspring generated (Offsprings) generated per reference population evaluations and the probability (Probability) that a repaired candidate solution will replace its parent in the search population. Individually, the first factor was not significant while the second was extremely significant. As above, the interaction between the the two factors is insignificant.

Analysis 12 considers the probability of replacement (Probability) and the choice of repair method (Repair). Individually, both of these factors is extremely significant. Surprisingly, the interaction between the two factors is insignificant.

Table 20 shows the parameters and results for the 5 best experiments. Their trajectory is shown in Figure 30. Notice that the path of the best result, # 95, has the steepest initial decent, then continues to make modest gains. It would be interesting to visually observe the incremental changes in the conformation during the run. However, it this time it is not possible as only the initial and final conformations are output.

While the analysis of this experiment set provides insight into the exogenous parameters, the conclusion can only be taken so far. A stronger conclusion can be made for the parameters with binary values, (Ref Point and Repair) than for the others whose value were heuristically selected. In particular, the system is sensitive to the replacement probability, additional values should be examined.

5.4 Experiment IV: Analysis of Para-REGAL

The experiments detailed in Section 4.5.3 are summarized in Table 21. The best results were obtained when both $P_m = P_{cm} = 0.66$. Interestingly, in this experiment, the "best" results were not obtained with the tighter constraints. That is, the best candidate was never found in *Island 3* which is where the tight constraints were active. One possible explanation is that it is more difficult to generate fully feasible

Table 20. 5 Best Results Exogenous Parameter Analysis

Exp	Ref	Search	Periodic	Offsprings	Prob	Ref Point	Repair	Fitness	Run Time
95	20	50	50	20	0.5	1	0	-31.98	4.56
282	50	20	100	30	0.2	0	1	-31.25	5.28
285	50	20	100	30	0.5	0	0	-31.11	3.59
348	50	50	50	20	0.5	1	1	-30.65	5.11
492	50	70	100	20	0.5	1	1	-29.94	4.72

Reference Point (0=random, 1=ordered)
Repair Method (0=random, 1=deterministic)
Run Time is in hours

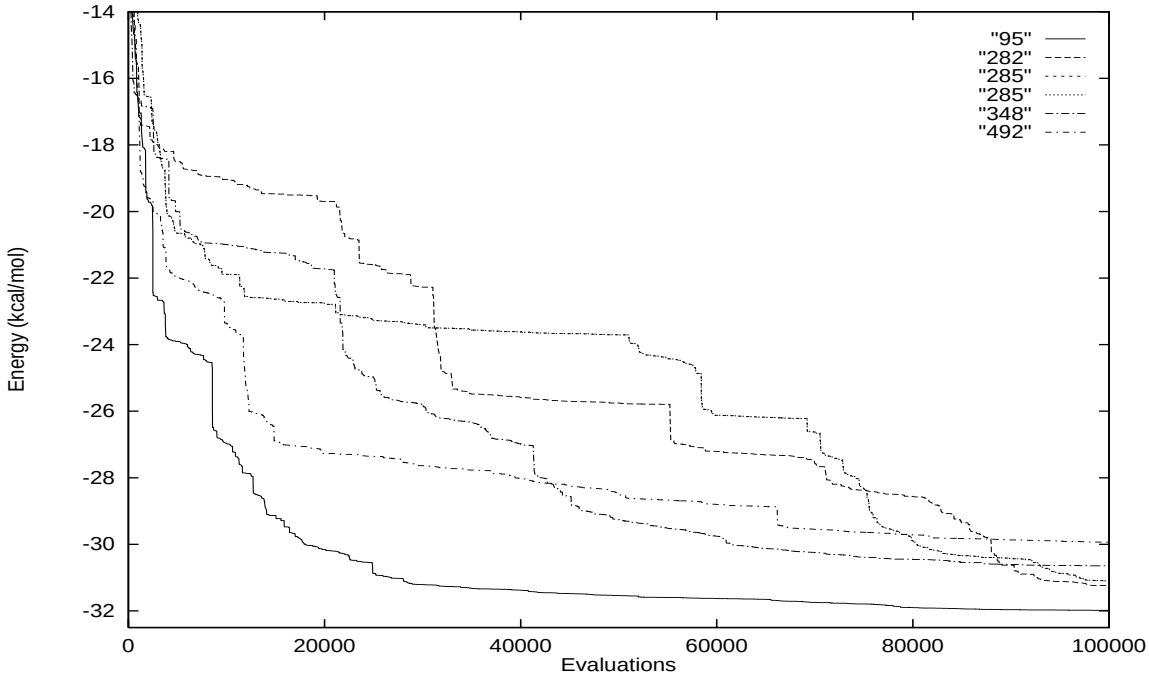


Figure 30. Trajectory Plot, 5 Best Phase I Experiments

candidates using the tighter constraints. With fewer candidates generated, there is simple less exploration of the search space.

A candidate feasible for the tight constraints is also feasible, in most cases, for the relaxed constraints, but nor the reverse. A update to the more restrictive subpopulation will be accepted and acted upon by the more relaxed subpopulation. But the update to the more relaxed subpopulation is likely to not be admissible into the more restrictive population, thus limiting the diversity in the restricted subpopulation. Another factor could be the exogenous parameter *periodicity* (the period at which the reference population is explicitly evolved). A value of 250 was used in this set of experiments which is larger than that studied in Experiment III.

One final observation from this experiment, it does not appear that taking m evaluations and distributing them over n processors ($\frac{m}{n}$ per island) produces the same result as m serial evaluations. Examination of trajectory shows the improvement in the energy function to be very fine; frequently the improvement is on the order of 10^{-3} cal/mol. That is, the improvement is in the $6th$ decimal place. This small step size is a factor of the real-valued representation and the ruggedness of the fitness function. Therefore, it is believed that a depth in the number of evaluation, on the order of $50 - 100K$ is required to achieve desired results.

Table 21. Para-REGAL Results with Four Islands

Experiment Number	P_m	P_{cm}	Average Fitness	Minimum Fitness	Island(s) of Best Solution
1	0.00	0.00	-21.81	-22.38	0,1,2
2	0.00	0.33	-21.81	-22.38	0,1,2
3	0.00	0.66	-21.81	-22.38	0,1,2
4	0.00	1.00	-21.81	-22.38	0,1,2
5	0.33	0.00	-20.57	-21.62	0,1
6	0.33	0.33	-23.35	-23.45	0,1
7	0.33	0.66	-23.13	-23.93	0,2
8	0.33	1.00	-21.31	-21.52	0,2
9	0.66	0.00	-21.76	-21.92	0,2
10	0.66	0.33	-22.12	-22.71	2
11	0.66	0.66	-25.78	-25.79	0,1,2
12	0.66	1.00	-23.31	-24.24	0,2
13	1.00	0.00	-21.80	-23.35	0,1
14	1.00	0.33	-24.49	-25.16	2
15	1.00	0.66	-22.49	-22.49	0,2
16	1.00	1.00	-22.62	-22.67	0,2

5.5 Summary

The experiment sets show beneficial result from the approaches proposed in this research. While the hybrid GA had been the most effective technique applied to PSP using CHARMM energy model, it requires considerable resources[5] In Experiment I, the parallel farming model is more efficient (faster), considerably reducing wall clock time while producing identical results. However, there are upper bounds on increases realized by this method, because as the ratio of processors to population size increases, idle time increases.

In Experiment II, results using a real-valued GA implementation, REGAL, verify the feasibility of using domain knowledge to limit the search . Results are better when "good" domain constraints are included. The domain constraints considered in the second experiment set are of low fidelity. An experienced biochemist could certainly develop more precise sets. Of course, the constraints could be so tight that the only member

[5] Considerable resources is a relative concept. Even a 1000 hours of computer time is trivial when compared to experimental techniques that require years to yield results.

of the feasible search space is the optimal solution. But, if they can be defined that preciously, the protein folding problem would are ready be solved!

The GA community has less experience with exogenous parameter selection in real valued GAs, especially, GENOCOP-III. Therefore, the results of Experiment III are important. From an effectiveness perspective, the size of the reference population matters, as does the probability of replacement and how repair is performed. At least this is the case with respect to PSP.

The results of experiments on the parallel version of REGAL, Para-REGAL, reinforce the difference usually observed between serial and parallel GAs. In this case, better trajectories were observed in islands with more relaxed constraints. The next chapter presents overall conclusions and presents issues for further study from this research.

VI. Conclusions and Recommendation

This investigation has consumed thousands of hours of computer time conducting several thousand individual experiments. Additionally, hundreds of hours were spent reviewing the literature and designing and implementing software. These efforts are of secondary importance as related to the important contributions of conception, synthesis, design, engineering, and refinement of several significant solutions to a particular *Grand Challenge* problem. These accomplishments would not have been possible without insights drawn from multidisciplinary sources. The efforts are grouped into four distinct initiatives, each of which is discussed in turn.

6.1 Initiative I: PHGA

The Parallel Hybrid GA (PHGA) initiative enhances the performance of previous AGCT efforts with hybrid genetic algorithms by implementing a novel parallel architecture. Based on the farming model, PHGA preserves the effectiveness of the hybrid GA while reducing the biochemical researcher's turn around time. Built with the MPI standard rather than a proprietary communications library, the resulting code is reusable on a variety of high performance platforms and architectures without any source code changes. Thus, the researcher can utilize what ever computing resource are available and can easily migrate when new capacity becomes available.

The implementation is scalable to a number of processors bounded by the population size. However, past a threshold approximated by half the population, the implementation is not cost-optimal, and efficiency, in terms of processor utilization, decreases. Still, this is a powerful tool for PSP!

6.2 Initiative II: REGAL

The REal-valued Genetic Algorithm, Limited by constraints (REGAL), is a strong method for predicting molecular structures. This strength allows a biochemistry researcher to specify as much or as little about a molecular system as is known. Performance, effectiveness for a given level of effort, is proportional to the amount of domain knowledge included. The method of capturing domain constraints is generalized to a form that handles feasible regions including area from both sides of the *trans* orientation simultaneously. Thus, the method can be extended to dynamically adjust the constraints during execution.[1] For static constraints not involving the *trans* orientation, computationally more efficient linear methods can be used.[2]

In this experiment set, it was noted that the binary GA was less effective on the larger Polyalanine molecule, regardless whether local minimization was used or not. This behavior can be explained using the

[1] Actual implementation is out of scope for this investigation because research is required into appropriate control metrics.
[2] Limit the dihedral angle's range to a lower bound greater than $-\pi$ and an upper bound less than π.

fundamental theorem of genetic algorithms because the molecular representation used causes the "building blocks" to be longer in Polyalanine. The theorem (41) shows that the number of shorter, not longer, building blocks increase exponentially in subsequent generations. In REGAL, this tendency is overcome by constraints.

6.3 Initiative III: Examination of Exogenous Parameters

Considerable research has been applied to the determining optimal exogenous parameters values, i.e. crossover rate, mutation rate, for classical binary genetic algorithms. Less data is available for real-valued GAs, especially GENOCOP-III. Optimal parameters for applications based on GENOCOP-III are especially hard to determine because it incorporates the best of many Evolutionary Computation disciplines. While the results for this initiative are with respect to a specific problem (PSP), other GA researcher may find them insightful for problems characterized by high dimensional rugged fitness landscapes. The greatest single insight is the superiority of a random verses deterministic choice of α when repairing infeasible candidate solution by means of a linear combination with some reference point.

6.4 Initiative IV: Para-REGAL

Most parallel GAs perform better than their serial counterparts simply because the collective population undergoes more operations. Taken to extremes, such an algorithm becomes a exhaustive search of the solution space that yields the optimal solution. By giving the researcher an option to impose different constraints at each island, Para-REGAL more closely matches the hierarchical folding observed in proteins during *molten globular* states. That is, the constraints active at a particular processor, could be defined to, say, exploit the knowledge the a specific subsequence forms, say a, α-helix. A novel probabilistic migration concept balances exploitation of a specific region of the search space with exploration of the entire search space. Probabilistic migration also balances the algorithm with the message capacity of the underlying parallel or distributed architecture.

In Para-REGAL, assuming an adequately high probability of migration, each node maintains a near real time view of its progress verses the progress of other nodes. This knowledge has the potential to dynamically effect system behavior by modifying constraints or other control parameters.

The contribution of each of these initiatives to the AGCT GA Toolkit is shown in Figure 31.

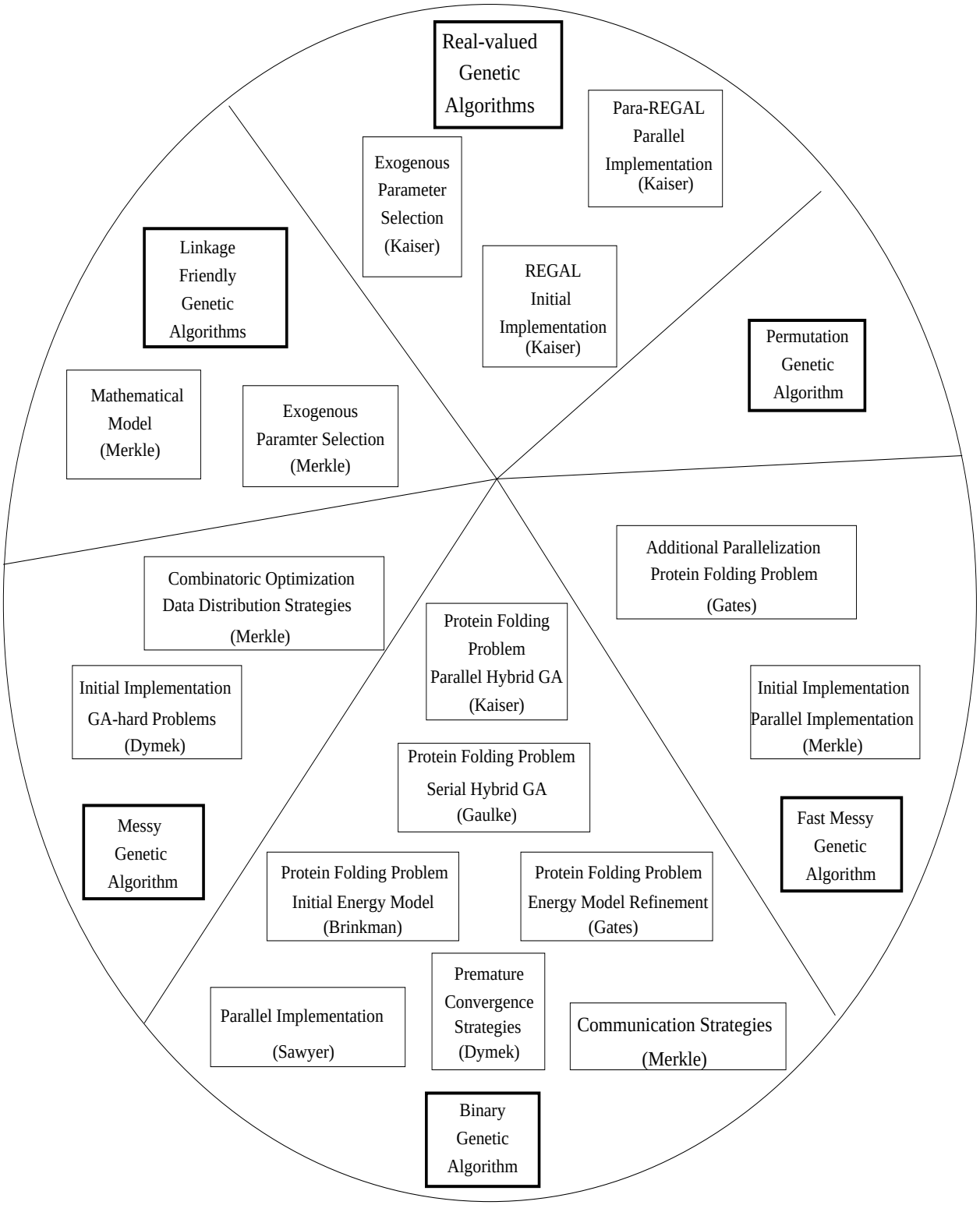


Figure 31. AGCT's Genetic Algorithm Toolkit (Including this Thesis)

6.5 Recommendations

The success of using binary encoded and real-valued GAs for the two proteins suggests their application to more complex protein folding problems. Currently, preparing the inputs in the format acceptable to the current implementations is manual and error prone. During the Summer of 1996, interns worked very hard to document, streamline, and possibly automate the preparation process. In spite of this effort, preparing a new molecule for study is difficult. Perhaps this is the incorrect approach. The molecular data structures used in the current implementations have a number of flaws. The first is a dependency on a specific atom ordering. This ordering is not consistent with the commercial software packages used to generate the amino acid sequences representing the molecule in question. Unfortunately, there is sufficient variability in the behavior of the commercial packages that a transform process is not feasible. Separately, the current data structures do not have methods for hierarchical understanding of the system in question. The representation is strictly at the atom level, there is no concept of composite objects such as residues, sequences, or other molecules. Much of the domain knowledge available of exploration is in terms of these composite objects. Therefore, I recommend that effort be applied to the development of an object oriented molecular model.

In applying GAs to more and more complex proteins, the use of constraints may be the only way of obtaining acceptable solutions due to the exponentially increasing number of local and global optimal. The constraints used in the research were developed manually. There is a wealth of knowledge to be discovered from sources such as the Brookhaven Protein Data Bank. There is a need for data mining tools to classify knowledge from which to form constraints. Development of such tools should include researchers from both the AI and biochemistry fields.

My characterization of the exogenous input parameters with respect to REGAL was the first step. The question of optimal operator frequency was deferred by using the adaptive option. Now that there is insight into optimal values for the input parameters, a study of the operators is in order. In addition, local minimization which is effective with a binary encoding, should be even more effective in a real-valued encoding. Initial results suggest local minimization causes the execution to become trapped early-on in a suboptimal minima. An interesting approach would use local minimization as an operator.

6.6 Summary

This chapter shows my research following two separate GA tracks. The first extends previous research in binary GAs in a significant way. The second track has been into virgin territory, at least as far as molecular structure predication is concerned. The result, I think, is a quantum jump in the applicability of evolutionary computation as a solution to a problem of large dimension. The recommendations indicate additional graduate research which should result in substantial contributions to DOD computational technology.

Appendix A. Background on the Protein Folding and Protein Structure Prediction Problems

This appendix contains background material on the protein folding and protein (or polypeptide) structure prediction problems, most of which has been presented in previous AFIT theses, particular Brinkman (5) and Gates (35). Section A.1 defines terminology in the biochemistry domain. Section A.2 describes the expensive experimental techniques used to determine the structure of proteins. Finally, Section A.3 examines various models used to predict the structure of polypeptides and proteins.

The protein folding problem (PFP) has been recognized as a National Grand Challenge problem in biochemistry and high-performance computing (11). The challenge is to find a method to predict the three-dimensional topology of a protein based on the sequence of its components. A solution, which would provide knowledge about the function(s) of individual proteins, is also the first step toward solving the *inverse folding problem* (IPFP) (8, 71). The inverse folding problem is to determine a sequence (possibly more than one) that fold to a specified three-dimensional structure.

The difference between the two problems is best characterized by the ability a solution to either would provide: a PFP solution would enable the *evaluation* of many proteins in a search for one with a specific property or function; an IPFP solution would provide a mechanism to *design* a protein with specified characteristics (8:25–26). Possible applications include: pharmaceuticals with few or no side effects; energy conversion and storage capabilities (similar to photosynthesis); biological and chemical catalysts and regulators; angstrom scale information storage; and possible optical/chemical shielding from harmful radiation sources (8:25) (71:5) (93).

A.1 Introduction to Proteins and Associated Terminology

Proteins (polypeptides) are linear sequences of the 20 naturally occurring amino acids. Each amino acid consists primarily of three common *backbone* atoms (a nitrogen and two carbons [N$-$C$_\alpha$-C$_\gamma$]) and a distinct combination of atoms and covalent bonds, called the *side-chain* (S$_i$), connected to the C$_\alpha$ carbon atom. A particular protein is defined by a unique amino acid sequence known as the *primary structure* of the protein (8:24)(71:2)(69:49).

As the amino acids form into proteins via peptide bonds, they give up a water molecule. The linked amino acids are called *residues*. Figure 32 depicts a generic protein composed of three residues (amino acids). In most context, the terms amino acid and residue are used interchangeably. The primary structures of approximately 50,000 naturally occurring proteins are currently known and this number is expected to double every year, due largely to the Human Genome Project and the ease with which sequences are experimentally determined (71:5)(91). In fact, the sequence determination and also fabrication is fully automated.

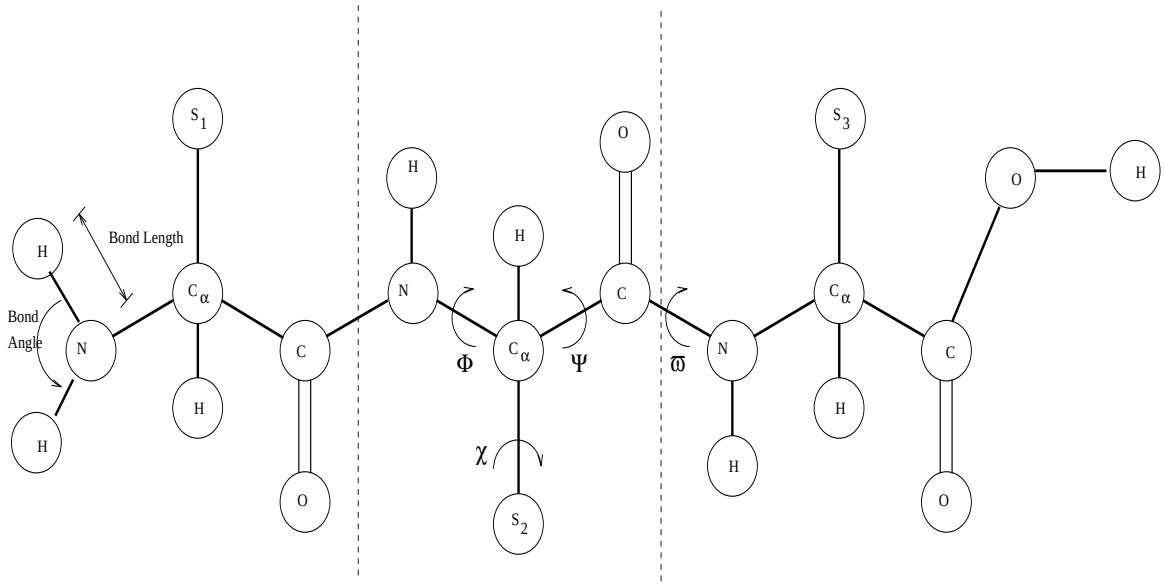


Figure 32. A Three Amino Acid Protein

Subsequences of proteins tend to exhibit regular patterns. Two common patterns are *α-helices* and *β-sheets*. These describe the *secondary structure* of a protein (8:24). Secondary structures result only when at least four or five consecutive amino acid residues have similar ϕ and ψ values (57). Some researchers are investigating the utility of predicting secondary structure as the first step of tertiary structure prediction (69:50). This technique has had limited success. The problem is that even though certain residues are found more frequently in specific secondary structure, the greatest preference is only twice that of other secondary structures. In most cases, the preference is much smaller (108:422). Table 22 identifies the values for (ϕ, ψ) angle pairs that according to Horton (57) ideally define commonly occurring secondary structures.

Table 22. ϕ, ψ Pairs of Common Secondary Structures

Secondary Structure	phi(ϕ)	psi(ψ)
α helix (right handed)	-57	-47
α helix (left handed)	57	47
3_{10} helix (right handed)	-49	-26
Antiparallel β sheet	-139	235
Parallel β sheet	-119	113
Collagen helix	-51	153
Type II turn (second residue)	-60	120
Type II turn (third residue)	90	0
Fully extended chain	-180	-180

The three-dimensional structure of a protein is the major determinant of its function. This three-dimensional shape is called the *tertiary structure* or *conformation* of the protein. Proteins assume their

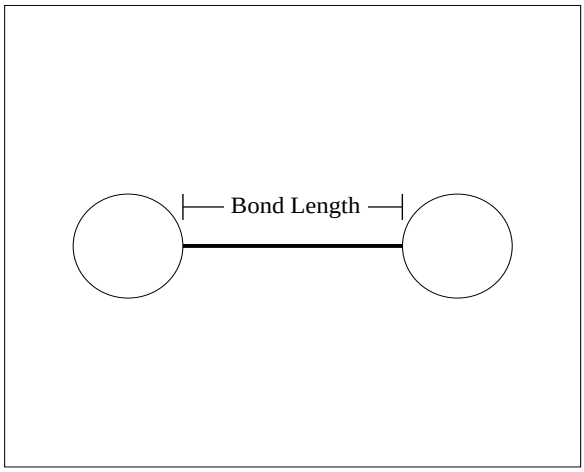


Figure 33. Protein Bond Length

native conformation, which is unique and compact, in their natural biological environment (typically in aqueous solution, at neutral pH and 20–40° C) (8, 71). A protein in its native conformation is only slightly more stable than the various conformations with marginally higher energies. Normally, there is only a 10 kcal/mol energy difference between the completely folded and unfolded conformations. This single fact is responsible for the major difficulty of the protein folding problem (8:24–25) (71:2–4) (69:50).

There are two principle coordinate systems used to identify the position of the atoms in a molecule. The Cartesian coordinate system uses a three dimensional coordinate (x_i, y_i, z_i) , $1 \leq i \leq n$, where n is the number of atoms in the molecule. An arbitrary atom, usually C_{α_1} is assigned to the origin. This system is most useful to compute the distance, $d_{ij} = \sqrt{(x_i - x_j) + (y_i - y_j) + (z_i - z_j)}$, between two atoms. With this system each molecule has $3n$ degrees of freedom.

Internal coordinates is the other coordinate system. The dihedral angle approach defines position of all atoms in a protein from the position of one atom (usually at the origin), the *bond length* of each covalently bonded pair of atoms, the *bond angle* formed by each triplet of bonded atoms, and the *dihedral angle* formed by each bonded group of four atoms (see Figures 33 - 35). Given this set of parameters, every protein has $3n - 6$ degrees of freedom where n is the number of atoms. However, the bonds and bond angles are relatively rigid, therefore the independent dihedral angles are left as the only dominant factor to determine the tertiary structure of a protein and the degrees of freedom are reduced by a factor of approximately 2/3 (8:26) (69:50).

Each amino acid contains a ϕ, ψ, and ω dihedral angles and zero or more χ_i dihedral angles as shown in Figure 32.

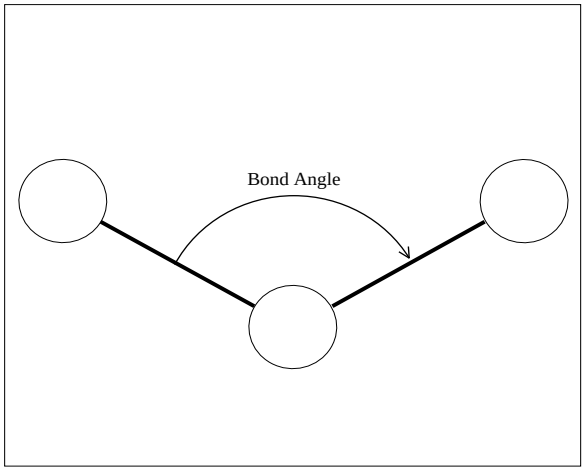


Figure 34. Protein Bond Angle

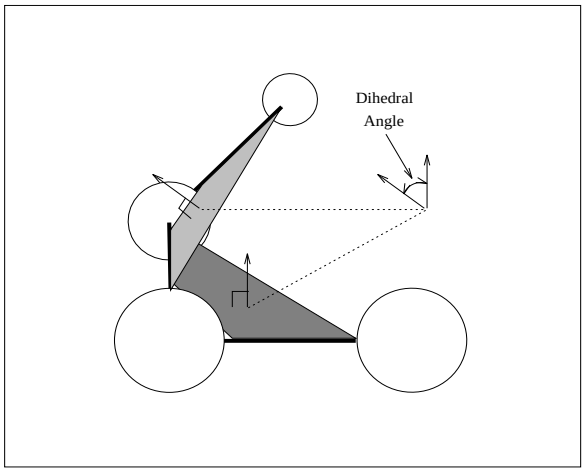


Figure 35. Protein Dihedral Angle

If we discretize the domain of the dihedral angles so that there are d possible values, then the size of the search space is given by d^N where N is the number of independently variable dihedral angles. Given a very coarse $20°$ discretization of the $0 - 360°$ dihedral angle domain and a small protein with 24 independently variable dihedral angles, the search space contains $18^{24} \approx 1.3 \times 10^{30}$ conformations. Table 23 shows the time required to enumerate the search space on current and envisioned high performance computers (under the optimistic assumption of one evaluation per clock cycle)(107:7)! (Giga-, Tera-, and Peta-FLOP computers perform $10^9, 10^{12}$, and 10^{15} floating point operations per second, respectively) Therefore, if we hope to find the single native conformation of a protein, we must have access to efficient search algorithms that severely prune the search space.

Table 23. Enumeration Time of a 1.3×10^{30} Search Space at One Solution per Clock Cycle

Computer Speed	Execution Time (years)
1 GigaFLOP	$\approx$ 41 trillion
1 TeraFLOP	$\approx$ 41 billion
1 PetaFLOP	$\approx$ 41 million

A.2 Experimental Tertiary Structure Determination

In comparison with the number of known protein sequences, the number of known native conformations at high resolution is extremely small (less than 4000). The tertiary structures of these proteins have been determined experimentally using either X-ray crystallography or nuclear-magnetic-resonance (NMR) spectroscopy. These techniques are inadequate for the task because they take up to several years to obtain results for a single protein (8:25)(71:5). Thus, the quest is on for a method to accurately predict the structure without actually observing it. Even if this is not accomplished, a method that predicates the position of the backbone atoms can reduce the time required for experimentally determination to less than a year.

A.3 Tertiary Structure Prediction (PFP)

To reduce the gap between the number of known protein sequences and native conformations, we need to be able to reliably predict the tertiary structure of proteins in a reasonable amount of time. *Exact* versions of the classical methods discussed next are theoretically capable of finding the native tertiary structure of any protein. In practice, the computational cost of the calculations prohibits the use of these exact methods. The classical methods that are computationally viable are typically relaxed formulations that ignore the high-order interaction terms. The applicability of the other prediction methods discussed below is severely limited.

A.3.1 Classical Prediction Methods. *Molecular dynamics* is a technique that attempts to simulate the protein folding process. The protein is treated as an N-body simulation and Newton's motion equations are solved to determine the location of all the atoms at discrete points in time. Molecular dynamics faces two major difficulties in its attempt to fold proteins. First, the number of atoms that must be simulated is very large:

1. Small proteins contain hundreds of atoms.
2. Larger proteins can be composed of several ten-thousands of atoms.
3. Thousands of atoms must be added to simulate the surrounding solution.

Second, the thermal oscillations of bonded atoms have a period between $10^{-14} - 10^{-13}$ seconds. Simulation time steps in the femtosecond (10^{-15} sec) range are required to accurately account for these harmonics. These two factors have limited molecular dynamics simulations to less than a few nanoseconds (10^{-9} sec), even on today's fastest supercomputers. That time-frame is ten orders of magnitude too short to simulate the folding process of most proteins (8:27)(71:6–7). Using an *extended-atom representation* is one method that can greatly reduce the impact of these two problems. The extended-atom representation combines hydrogen atoms with the heavier atoms they are bonded to. This representation generally halves the number of "atoms" in the problem and allows the size of the simulation time steps to be increased (6:189).

The *energy minimization* approach assumes that proteins, like other physical systems, assume that state which minimizes total energy in the system (however, this assumption is not universally accepted (58)). There are three types of energy minimization methods that differ by their time complexity and the accuracy of their calculations. *Ab initio* methods calculate the energy exactly. *Semi-empirical* methods eliminate the non-dominating interaction integrals from the calculation. *Force-field* methods simply account for the pairwise interactions between atoms with an appropriate parameterization that implicitly accounts for multi-particle interactions (71:6). Table 24 compares the time complexity of these three energy minimization models and gives example execution times for a moderately sized protein ($n = 1000$) assuming the individual component calculations take one nanosecond (10^{-9} sec).

Table 24. Time Complexity of Energy Minimization Methods

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

CHARMM and ECEPP are examples of force field energy models. The ECEPP/2 energy model is shown in Figure 36. Its four terms represent the energy due to dihedral angle deformation, non-bonded interactions, electrostatic interactions, and hydrogen bond energy respectively. ECEPP is the most widely

$$
\begin{aligned}
E = \quad & \sum_{(i,j,k,l)\in\mathcal{D}} \left(\frac{U_{0_{ijkl}}}{2}\right)(1 \pm \cos(n_{ijkl}\Theta_{ijkl})) + \\
& \sum_{(i,j)\in\mathcal{N}} \epsilon_{ij}\left[F_{ij}\left(\frac{r_0}{r_{ij}}\right)^{12} - 2.0\left(\frac{r_0}{r_{ij}}\right)^{6}\right] + \\
& \sum_{(i,j)\in\mathcal{N}} \left[332.0\left(\frac{q_i q_j}{D r_{ij}}\right)\right] + \\
& \sum_{(i,j)\in\mathcal{H}} \epsilon_{ij}\left[\left(\frac{r_0}{r_{HX}}\right)^{12} - 2.0\left(\frac{r_0}{r_{HX}}\right)^{10}\right]
\end{aligned}
\tag{15}
$$

Where

- $\mathcal{D}$ is the set of 4-tuples defining ω and χ dihedrals,
- $\mathcal{N}$ is the set of non-bonded atom pairs,
- $\mathcal{H}$ is the set of hydrogen bonding atom pairs,
- r_{HX} is the donor-acceptor distance,
- r_{ij} is the distance between atoms i and j,
- Θ_{ijkl} is the dihedral formed by atoms i, j, k, and l,
- q_i is the partial atomic charges of atom i,
- the $U_{0_{ijkl}}$'s, n_{ijkl}'s, F_{ij}'s, ϵ_{ij}'s, r_0's, and D are empirically determined constants.

Figure 36. ECEPP/2 Energy Model as Implemented by AGCT

used energy model in PSP research. Our initial test molecule, Met-enkephalin, the accepted native structure has been identified by minimizations on the ECEPP model. It is a polypeptide specific energy model, thus it has limited utility for Wright Labs research into novel materials. The CHARMM (6) energy model is shown in Figure 37. The five terms (which we denote $E_{\mathcal{B}}$, $E_{\mathcal{A}}$, $E_{\mathcal{D}}$, $E_{\mathcal{N}}$, $E_{\mathcal{N}'}$) represent the energy due to bond stretching, bond angle deformation, dihedral angle deformation, non-bonded interactions, and 1-4 interactions, respectively. Other terms are available but are not implemented because their contributions are insignificant.

A.3.2 Other Prediction Methods. Structure prediction by *homology* attempts to align the sequence of a protein with an unknown tertiary structure with one whose native conformation is known (71). It has been observed that if the sequences are similar, then the conformations are frequently nearly identical. An extension of homology, called *sequence-structure alignment*, builds a partial monotonic mapping directly from the sequence of the unknown protein to the known tertiary structure of the similar protein. The differences between the two structures are usually surface characteristics built upon the same core structure. Both of

$$
\begin{aligned}
E = \quad & \sum_{(i,j)\in\mathcal{B}} K_{r_{ij}}(r_{ij} - r_{eq})^2 + \\
& \sum_{(i,j,k)\in\mathcal{A}} K_{\Theta_{ijk}}(\Theta_{ijk} - \Theta_{eq})^2 + \\
& \sum_{(i,j,k,l)\in\mathcal{D}} K_{\Phi_{ijkl}}[1 + \cos(n_{ijkl}\Phi_{ijkl} - \gamma_{ijkl})] + \\
& \sum_{(i,j)\in\mathcal{N}} \left[\left(\frac{A_{ij}}{r_{ij}}\right)^{12} - \left(\frac{B_{ij}}{r_{ij}}\right)^{6} + \frac{q_i q_j}{4\pi\varepsilon r_{ij}} \right] + \\
& \frac{1}{2} \sum_{(i,j)\in\mathcal{N}'} \left[\left(\frac{A_{ij}}{r_{ij}}\right)^{12} - \left(\frac{B_{ij}}{r_{ij}}\right)^{6} + \frac{q_i q_j}{4\pi\varepsilon r_{ij}} \right]
\end{aligned}
\tag{16}
$$

Where

- $\mathcal{B}$ is the set of bonded atom pairs,
- $\mathcal{A}$ is the set of atom triples defining bond angles,
- $\mathcal{D}$ is the set of atom 4-tuples defining dihedral angles,
- $\mathcal{N}$ is the set of non-bonded atom pairs,
- $\mathcal{N}'$ is the set of 1-4 interaction pairs,
- r_{ij} is the distance between atoms i and j,
- Θ_{ijk} is the angle formed by atoms i, j, and k,
- Φ_{ijkl} is the dihedral angle formed by atoms i, j, k, and l,
- q_i is the partial atomic charges of atom i,
- the $K_{r_{ij}}$'s, r_{eq}'s, $K_{\Theta_{ijk}}$'s, Θ_{eq}'s, $K_{\Phi_{ijkl}}$'s, γ_{ijkl}'s, A_{ij}'s, B_{ij}'s, and ε are empirically determined constants (taken from the QUANTA parameter files).

Figure 37. CHARMM Energy Model as Implemented by AGCT

these methods are severely limited by our tiny database of currently known protein structures. They are also incapable of predicting the native conformation of proteins with novel structures (71:7–9).

Simplification techniques are used as methods to reduce the conformational search space to a size that will to today's algorithmic search strategies (8). *Lattice* models reduce three-dimensional space to a structured grid, where atoms can only be placed on the grid points. The grid is designed to accommodate the typical connections observed in real proteins. In such a discrete search space, the protein can be modeled as a walk on the lattice points, thus allowing for classical search techniques. The disadvantage is an obvious lose of fidelity by attempting to represent a continuous domain with a discrete approximation. Further simplifications have been obtained by reducing or eliminating the explicit representation of side-chains (71:9–10). Dropping the side-chains will reduce the number of variable by up to one half. The optimal "fold" can be defined with just the backbone representation. But the minimal energy conformation must also consider contributions from the side chain.

Appendix B. Background on Genetic Algorithms

This appendix is a background discussion of genetic algorithms (GAs). Most material has appeared in previously AFIT theses, Brinkman (5), Dymek (23), Gates (35), and Merkle (77). Section B.2 provides a historical context for evolutionary algorithms. Sections B.3 through B.5 present the theory and mechanics of simple GAs (SGAs), messy GAs (mGAs), and fast messy GAs (fmGAs) respectively. Finally, Section 2.5 describes techniques used to parallelize genetic algorithms.

Genetic algorithms (GAs) are a stochastic search technique loosely based on natural evolution and the Darwinian concept of "Survival of the Fittest" (41:1)(56). A generalized genetic algorithm consists of a *population* of *encoded* solutions that are manipulated by a set of *operators* and evaluated by some *fitness function* that determines which solutions survive into the next *generation*.

For our purposes, search and optimization techniques fall into two broad categories: deterministic (a combination of calculus-based and enumerative) and stochastic (random) methods (41:2). Greedy algorithms, hill-climbing, calculus-based methods, and branch and bound tree/graph search techniques are all examples of deterministic approaches (4). These methods have been successfully used to solve a wide variety of problems. However, there is an even greater number of large dimensional problems that are discontinuous, multi-modal, or NP-complete where deterministic methods are ineffective (41:3–6)(31, 34). The main short coming for deterministic methods is their requirement for some amount of problem specific knowledge to direct or limit the search. For example:

1. Hill-climbing algorithms are limited unimodal functions
2. Greedy algorithms assume optimal sub-solutions are *always* part of the optimal solution (4:80)(65)
3. Calculus-based methods require continuity (68:167)
4. Branch and bound search techniques need problem specific heuristics/decision algorithms to limit the search space (34, 95)

A partial list of problem characteristics that can make deterministic search techniques unsuitable for a particular problem includes: multi-modal and/or discontinuous solution spaces, exponential search spaces (NP-complete problems), and limited domain knowledge (no heuristics). Problems that exhibit one or more of these characteristics are called *irregular* (67).

Stochastic search and optimization approaches (simulated annealing, evolutionary strategies, evolutionary programming, genetic algorithms, Monte-Carlo techniques) have been developed as an alternative to deterministic techniques (41, 82). The only requirements for stochastic methods are a function that assigns fitness values to possible solutions and an encode/decode between the algorithm and problem spaces. Although these methods cannot guarantee the optimum solution, in general they can provide *good* solutions to a wide range of problems that may be irregular and/or exponentially too large dimensionally for deterministic

methods (41:6–7)(65). Simulated annealing and Monte-Carlo techniques, in addition to GAs, are frequently applied to the PSP problem.

This appendix reviews the current literature on genetic algorithms starting with a short historical perspective of evolutionary algorithms in section B.2. Sections B.3, B.4, and B.5 discuss the rationale and mechanics of simple GAs, messy GAs, and fast messy GAs respectively. Finally, section 2.5 summarizes the current state of parallel genetic algorithms.

B.1 Brief History of Evolutionary Algorithms

Evolutionary models based on natural selection and genetic theory started appearing during the late 1950s and early 1960s. The first working models were computer simulations of genetic and biological systems. The idea of using algorithms that model natural evolution as search and optimization techniques began in the late 1960s and 1970s (41:89–104). The class of methods called evolutionary algorithms (EAs) is composed of three main categories based on that early work: Evolutionary Programming (EP), Genetic Algorithms (GAs), and *Evolutionstrategie* (Evolution Strategies, ESs) (3, 17) (41:104–106). However, this taxonomy is not universally accepted. Many authors categorize only EP and ESs as Evolutionary Algorithms, and put GAs in a separate category by themselves (30:24). Even though simulated annealing (SA) is based on thermodynamics, it's often associated with evolutionary algorithms because it uses a mutation operator (105:17). However, SA fail other "test" for inclusion with EA's because it operates on a single candidate solution rather than a population. Also difficult to categorize are hybrid combinations, such as GAs with deterministic local search or GAs with novel data representations. Such algorithms are simply called EAs (83).

Fogel, Owens, and Walsh first proposed the technique known as evolutionary programming. Evolutionary programming tries to generate computational biological evolution through a process that allows the survival of organisms that respond appropriately to a given environment. That is, it attempts to evolve a Finite State Machine (FSM). It has been applied to problems such as sequential symbol prediction and process control. EP usually operates on the components of abstractions such as finite state machines or programming languages (3) (31) (32).

Evolution strategies was conceived by Ingo Rechenberg and Hans-Paul Schwefel at the Technical University of Berlin while they were searching for optimal airfoil shapes. The original formulation consisted of a one-member population that was operated on by mutation only. The representation consisted of a pair of real-valued vectors ($V = \langle \mathbf{x}, \sigma \rangle$) where the first vector, $\mathbf{x}$, represents a solution and the second vector, σ, is a vector of standard deviations. The mutation operator creates a new individual at the t generation using a

normal distribution with zero mean as follows:

$$\mathbf{x}^{t+1} = \mathbf{x}^t + N(0, \sigma) \tag{17}$$

Various refinements have been made to this original design including population sizes greater than one, recombination operators, and dynamic changes to the vector σ (84:160–164)(3).

B.2 Origins of Genetic Algorithms

John Holland's work on adaptive systems is recognized as the fundamental beginning of genetic algorithms. Previous researchers had used computers to simulate evolutionary systems, and Fraser even tried to optimize a phenotype[1] function, but nobody before Holland recognized the role that nature's evolutionary process could play in search and optimization (41). The embarkation point for all GA research is Holland's *"Adaptation in Natural and Artificial Systems"* (55) which established the mathematical basis for GAs, including the then unnamed *Schema Theorem* or *Fundamental Theorem of Genetic Algorithms*, discussed in Section B.3.3, and generalized schemes for reproduction, crossover, mutation, and inversion (41:89–92).

Three other names have come to be synonymous with GA research: Kenneth A. De Jong, David E. Goldberg, and John J. Grefenstette. De Jong's dissertation (18) put Holland's theory to the test and introduced GA infrastructure that is still in use today (a suite of test functions and performance measures). Although his dissertation focused on function optimization, most of his work to date concerns his broader interest in machine learning (103, 19, 20). Goldberg began by applying genetic algorithms to machine learning and optimization problems. His most recent efforts include work on optimal GA population sizes and alternate GA paradigms (messy GAs and fast messy GAs) to combat deceptive problems (41:387–389) (45, 43). Grefenstette is probably best known for his genetic algorithm implementation, GENESIS, which has been used as a basic GA workbench by many researchers, including those at AFIT (51). He has also worked on optimal GA parameter sets and machine learning using genetic algorithms (49, 53, 50, 52).

B.3 Simple Genetic Algorithm (SGA)

Simple genetic algorithms are based on theories of natural genetics and therefore share some of the same terminology. Figure 38 illustrates the following terms. A *string* or *chromosome* contains *genes* that encode a solution to a particular problem. A *locus* and *allele* are associated with the gene. The locus, or position of a gene generally determines problem association. A gene is given only one value or allele at a

[1] The term **phenotype** refers to the traits expressed by an individual, in this case the value returned by a function. Contrast this with **genotype** which refers to the traits that define the individual, for example the parameters of the function

time from a set of values (the alleles) allowed for that gene. A bag of strings is called a *population*. A genetic algorithm *evolves* populations toward better solutions of the encoded problem by generations.

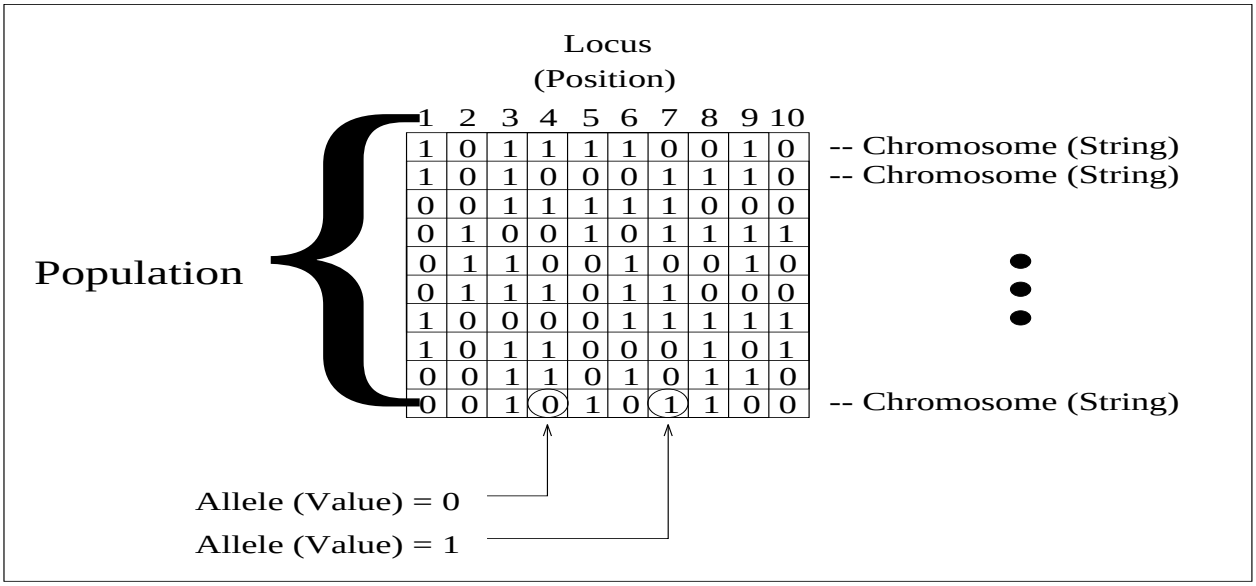


Figure 38. Simple Genetic Algorithm Data Structures and Terminology

Most SGA implementations use strictly binary encodings of the problem parameters, usually in a form such that x_{min} corresponds to a string of all 0's, x_{max} corresponds to a string of all 1's and there is a linear mapping of all values between x_{min} and x_{max}. Some problems may benefit from the use of Gray code parameter encoding (41:101). In Gray code, the encoding of successive integers differ by a single bit (66:40). It improves a mutation operator's chance of making incremental improvements, thus increasing the exploitation of favorable portions of the search space. Higher cardinality encodings have also been investigated for certain other problems, most notably combinatoric problems (92, 106). The problem encoding is a very important design decision in the formulation of a genetic algorithm to solve a specific problem. Binary offers the greatest domain independence while other representations are often more "natural" for, but limited to, a specific problem.

B.3.1 Simple Genetic Algorithm Operators. The three standard operators associated with simple genetic algorithms are *selection*, *crossover*, and *mutation* (41, 84). Above-average individuals of a population are *selected* to become members of the next generation more often than below-average individuals. *Crossover* recombines pieces of solutions to test different combinations of existing solutions. In the absence of other operators, selection and crossover will eventually force a population of solutions to *converge* to a single solution (41:14) (47). *Mutation* is an operator designed to encourage diversity in a population so that convergence occurs more slowly and more of the solution space can be explored.

Figures 39 and 40 illustrate the function of *single-point crossover* and *bitwise mutation* respectively on binary encoded strings that are ten bits long. Crossover operates on two strings, called the parents, to create two new strings, called the children. After two parents and a crossover point have been arbitrarily chosen, the bits after the crossover point are exchanged to create the children. Mutation is a unary operator that takes one string as input, arbitrarily chooses a bit position within the string, and changes the bit in that position to the opposite value. Many other crossover and mutation operators are possible, and indeed necessary, to exhibit different recombination characteristics and operate on alternate encodings (106, 112).

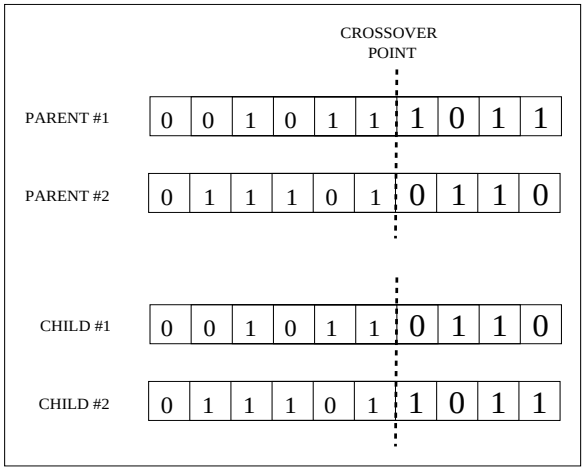


Figure 39. Single-Point Crossover

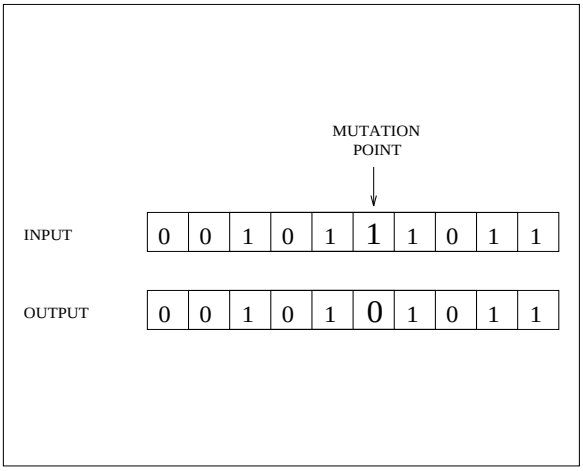


Figure 40. Bitwise Mutation

Figure 41 represents the operation of a *proportional* selection operator, called *roulette-wheel* selection, on two different populations of four strings each. Each string in the population is assigned a portion of the wheel proportional to the ratio of its fitness and the population average fitness. In the first case where the

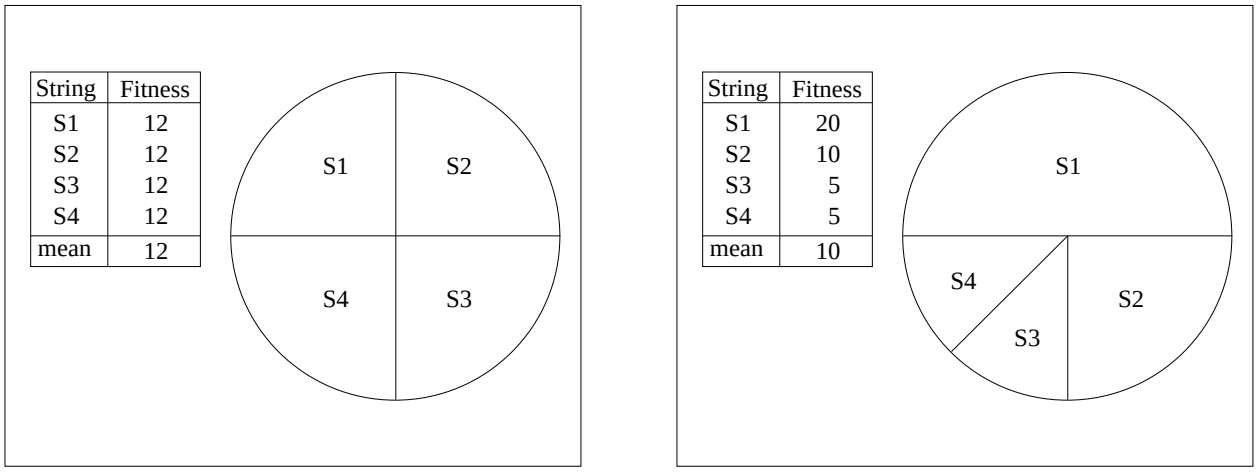


Figure 41. Roulette Wheel Selection

fitnesses are equal, each string is given an equal share of the wheel (it is equi-likely that any of the four strings is selected for the next generation). In the second example, $S1$ is twice as likely to be selected for the next generation as $S2$, which is twice as likely to be selected into the next generation as either $S3$ or $S4$. As with crossover and mutation, many other selection operator variations are possible, each with its own characteristic effect on convergence. *Rank-based* and *tournament* are notable selection operators, and the *elitist* strategy is a modification that can be used with any selection operator (41, 105, 110).

The three operators (crossover, mutation, and selection) and an evaluation function are assembled according to the pseudo algorithm shown in Figure 42 to create a simple genetic algorithm. Figure 43 is a graphic representation.

```
1.      randomly generate initial population
2.      evaluate fitness of all population members
        for i = 1 to the maximum number of generations and
                not some other stopping condition
3.          perform selection
4.          perform crossover
5.          perform mutation
6.          evaluate fitness of all population members
        end loop
```

Figure 42. Pseudo Algorithm for Simple GAs

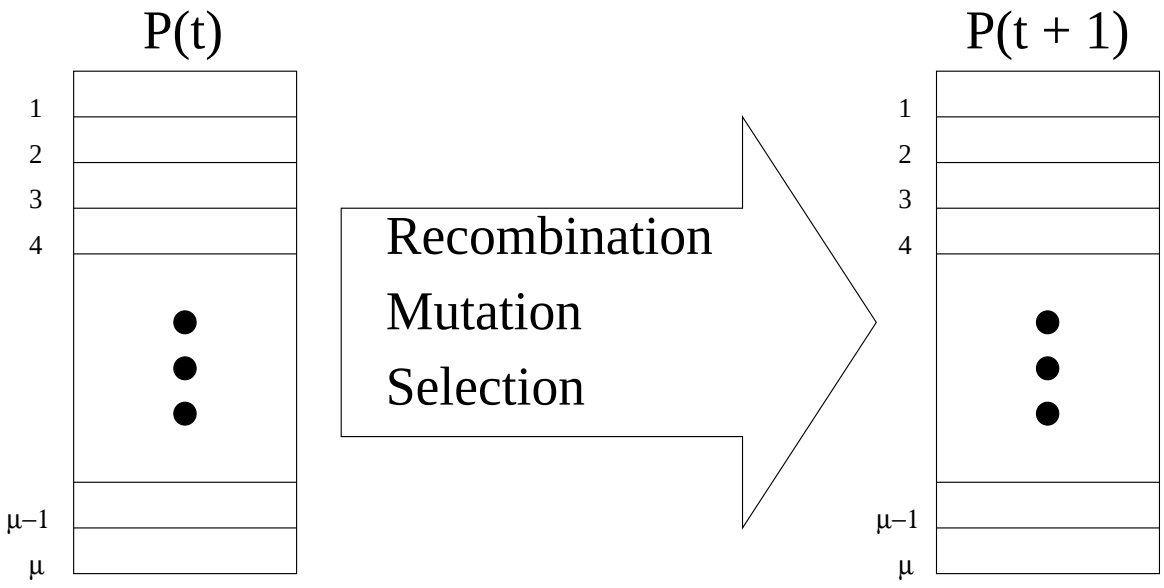


Figure 43. Simple Evolutionary Algorithm (GA)

B.3.2 Simple Genetic Algorithm Parameters. The most difficult part of genetic algorithms is selecting a parameter set that will generate the best performance (efficiency and effectiveness). Efficiency is a measure of the computer resources (cpu time, memory) required to obtain a solution. Effectiveness relates the solution quality of various algorithms. Both measures are relative to the specific problem being tackled and tradeoffs can be made between the two.

Some interrelated SGA parameters include: population size, crossover probability, and mutation probability. Theoretical analysis and empirical studies have been accomplished to formulate estimates of what each of these parameters should be to encourage a robust search that terminates with a near-optimal solution (42, 99). While these two particular studies seem at odds with each other, their conclusions are based on entirely different measurements of GA progress. Schaffer's empirical study is aimed at maximizing *on-line performance* or progress toward optimal solutions (efficiency) without regard for the final solution. Goldberg's theoretical work makes conservative choices to establish confidence levels for the optimality of a final solution (effectiveness) and ignores the astronomical resource costs required to achieve it.

Many relationships have been observed between parameter settings and performance. Increasing the population size generally improves the final solution, however the increase in execution time becomes prohibitive (18, 40). Population size has been shown to exhibit an inverse relationship with mutation rate and, to a lesser extent, crossover rate (99:55). Although there is no evidence so far of any correlation between crossover and mutation probabilities it is generally accepted that using crossover without mutation is insufficient for a robust search (25, 65, 100). However, it has been postulated (especially from the other branches of evolutionary algorithms) that mutation is the only necessary operator (30, 100). Other researchers are examining the effects of changing parameter settings during GA execution, either on some predefined schedule or possibly by monitoring GA metrics during the run (16, 28).

B.3.3 Mathematical Theory of How (Why) Simple GAs Work. *Schemata* are templates that define sets of strings with the same values at certain string positions and are represented using an additional *don't care* symbol (*) (41:19,29). For example, the schema *101 represents the set of strings {0101, 1101} and the schema 1*0*01 defines the set {100001, 100101, 110001, 110101}. The *defining length* ($\delta(H)$) and *order* ($o(H)$) are two values associated with a particular schema H. The defining length of a schema is a measure of the distance between the first and last fixed positions. The order of a schema is the number of positions with fixed values. Using the sample schemata from above $\delta(*101) = 4-2 = 2$, $o(*101) = 3$, $\delta(1*0*01) = 6-1 = 5$, and $o(1*0*01) = 4$.

The Schema Theorem, represented by

$$m(H,t+1) \geq m(H,t) \cdot \frac{f(H)}{\bar{f}} \left[1 - p_c \frac{\delta(H)}{l-1} - o(H)p_m\right], \tag{18}$$

establishes a lower bound on the number of representatives schema H will have in the next generation ($m(H,t+1)$) based on the:

1. number of representatives in the current generation ($m(H,t)$),
2. fitness of schema H vs the population average fitness ($\frac{f(H)}{\bar{f}}$),
3. string length, defining length, and probability that schema H will be destroyed by crossover ($p_c \frac{\delta(H)}{l-1}$), and
4. order and probability that schema H will be destroyed by mutation ($o(H)p_m$).

Although it would appear that genetic algorithms operate only on the specific strings in a population, it has been shown that many of the 2^l schemata in each string are processed simultaneously (*implicit parallelism* (55:71–72)(41:40)). The Fundamental Theorem of Genetic Algorithms (Schema Theorem) states that all schemata will receive representation in the next generation proportional to the ratio of their fitness to the average fitness of the population. This representation is reduced by the amount of disruption that crossover and mutation can cause to a schema. More succinctly,

> short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations (41:33).

B.3.3.1 Complexity Analysis. Using the standard SGA operators, the time complexity of selection, crossover, and mutation are generally $\mathcal{O}(nl)$ where n is the population size and l is the string length. Given a fixed number of generations, SGA execution time is $\mathcal{O}(nl)$ (21). However, the time complexity of the fitness function for real-world problems (expressed as some function of the string length and problem space parameters) usually dominates the time to execute the genetic algorithm control sequence (88), therefore great care should be taken in its analysis and design.

It is easy to see that the space complexity of a simple genetic algorithm is also $\mathcal{O}(nl)$ because the population of solutions has to be stored. However, the space complexity can also be affected by the data structure requirements of specific problems. For example, a function that relies on a lookup table to evaluate solutions requires more space. If that space is related to the problem size in a way that make the lookup table grow faster than the population size and string length, then the problem space requirements dominate the space complexity of the entire GA algorithm.

B.4 Messy Genetic Algorithm (mGA)

Genetic algorithms are designed to take advantage of the *building block* theory (33, 41, 55). The main idea is that small pieces of a solution which exhibit above average performance are combined to create larger pieces of above average quality, which are themselves recombined into larger pieces, and so forth.

Simple genetic algorithms suffer from the fact that the "pieces" that form the building blocks must be put next to each other explicitly in the fixed encoding or else they are more likely to be disrupted by crossover. This problem is magnified when competing schemata (schemata with different values at similar defining positions) define locally optimal solutions. *Deception* occurs when a locally optimal building blocks are selected instead of globally optimal ones. Messy genetic algorithms (mGAs) were designed to deal with these problems by encoding the string position (locus) along with its value (allele). This gives a messy genetic algorithm the ability to search for the "true" building blocks of the problem and create tighter *linkage* for those genes than a fixed position encoding would allow (46). The mGA encoding scheme also allows *under-specified* and *over-specified* strings to exist in the population. Under-specified strings don't have an allele defined for every locus and are evaluated with the aid of a locally optimal *competitive template* that supplies values for the unspecified genes. Over-specified strings contain multiple alleles specified at the same locus and are processed in a left-to-right fashion which sets the gene to the value encountered first. The desire to create and manipulate superior building blocks is the motivation behind messy genetic algorithms (45, 45, 44).

B.4.1 Messy Genetic Algorithm Operators. Messy GAs use variations of the same genetic operators used by simple GAs. In the few implementations of mGAs that exist (45, 46, 44, 77), tournament selection has been used instead of proportional or rank-based selection because of its desirable performance characteristics (45:50)(26, 42). The tournament selection operator also has a *thresholding* mechanism added to it which ensures that strings have a number of positions in common before competition is allowed (44:424–427). Crossover is replaced by a combined *cut-and-splice* operator that works on variable length strings. As the names suggest, *cut* divides a string into two smaller pieces and *splice* concatenates two strings to form a single, longer string. A mutation operator that can change a gene's value or its position has been described but unused in any mGA implementations (46:504).

Messy GAs employ a different initialization strategy compared to SGAs. The main processing loop of an mGA is composed of *primordial* and *juxtapositional* phases. During *partially enumerative initialization (PEI)*, exactly one copy of each possible building block of the specified size (k) is generated. Thus, the initial population size for a messy GA is generally quite large ($2^k \binom{l}{k}$) (46:420). The primordial phase serves two basic purposes: enrich the population with above average building blocks and reduce the population to a size that can be efficiently and effectively processed by the juxtapositional phase. Tournament selection, the

only active operator during the primordial phase, fills the population with above average building blocks, then periodically the population size is halved. No additional fitness evaluations are required during the primordial phase. The juxtapositional phase is most similar to the main processing loop of a simple GA (46:506). Cut-and-splice and any other genetic operators are applied, fitness evaluations are performed on the newly created strings, and tournament selection bolsters the next generation with highly fit solutions. A pseudo algorithm for messy GAs is shown in Figure 44.

```
1.    perform partially enumerative initialization
      evaluate fitness of all population members
2.    for i = 1 to the maximum number of primordial generations
          perform tournament selection
          if (a suitable number of generations have transpired) then
              reduce the population size
          end if
      end loop
3.    for i = 1 to the maximum number of juxtapositional generations
          perform cut-and-splice
          perform other operators (currently not used)
          evaluate fitness of all population members
          perform tournament selection
      end loop
```

Figure 44. Pseudo Algorithm for Messy GAs

B.4.2 Messy Genetic Algorithm Parameters. The major parameter settings associated with messy GAs are population size, cut-and-splice probabilities, and a schedule for reducing the population size. Initial population size can be calculated once the string length and block size have been determined. String length is simply a function of the encoding used, but block size is a problem dependent quantity that may be difficult to estimate. The final population size at the end of the primordial phase is even less quantifiable! The splice probability is consistently set to 1.0 with the following rationale: the primordial phase ends with a population of optimal building blocks which should only require assembly to form a complete string that is a near-optimal solution (45:25). The chosen cut probability is scaled by the current length of a string so that longer strings are more likely to be cut than shorter strings. The schedule for reducing population size during the primordial phase typically allows for two or three generations of enrichment followed by cutting the population in half (46:505). No theoretical or empirical work has been accomplished to provide any guidance for final primordial population size, cut probability, or population reduction schedules for messy GAs.

B.4.3 Mathematical Theory of How (Why) Messy GAs Work. The Schema Theorem (Equation 18) is directly applicable to messy genetic algorithms. The rationale for messy genetic algorithms follows from the theorem's interpretation: "short, low-order, above-average schemata receive exponentially increasing trials in subsequent generations." If the building blocks of a problem aren't encoded as short, low-order schemata then crossover and mutation will disrupt the formation of those building blocks.

For problems using a fixed encoding, where the identification of building blocks is prohibitive or impossible, Goldberg has calculated the *normalized expected defining length* ($\frac{\langle\delta\rangle}{l+1}$) for k-sized building blocks (Equation 19). The normalized expected defining length is a measure of the mean length of the schemata that make up the building blocks of a randomly encoded problem. The interpretation is that an arbitrary encoding is highly unlikely to establish tight linkage for the building blocks of a problem (46:498–499).

$$\frac{\langle\delta\rangle}{l+1} = \frac{k-1}{k+1} \tag{19}$$

Messy genetic algorithms take advantage of the Schema Theorem by searching for both the defining positions and gene values of the building blocks using PEI and the primordial phase. Then the juxtapositional phase of the messy GA starts with "short, low-order, above-average" schemata that are also "short, low-order, above-average" building blocks!

B.4.3.1 Complexity Analysis. Because of the partially enumerative initialization (PEI), the time complexity of messy GAs is $\mathcal{O}(l^k)$. This compares unfavorably with the rest of the algorithm which is only $\mathcal{O}(l \log l)$ (44:420–422). Space complexity remains unchanged from simple genetic algorithms. However, the constant term is generally larger and the population size (n) is *much* larger! As is the case with simple genetic algorithms, the time complexity of the evaluation function usually dominates that of the control sequence.

B.5 Fast Messy Genetic Algorithm (fmGA)

The advantage messy GAs have over simple GAs is the ability to create tightly linked building blocks for the optimization of deceptive problems. The disadvantage associated with this better processing is the time complexity of the initialization phase which dominates the mGA algorithm (44:422). Fast messy GAs are a messy GA variant designed to reduce the complexity of the initialization phase, and thus the overall algorithm time and space complexity (43:59).

B.5.1 Fast Messy Genetic Algorithm Operators. PEI and the selection-only primordial phase of mGAs are replaced by *probabilistically complete initialization* (PCI) and a primordial phase consisting of

selection and building block filtering (BBF) in fmGAs. PCI and BBF are an alternate means of providing the juxtapositional phase with highly fit building blocks (43:59–61).

PCI is used to create an initial population whose size is equivalent to the population size at the end of the primordial phase of mGAs. The length of these strings is typically set to $l - k$. The primordial phase then alternately 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). Building block filtering is a simple process that randomly deletes several genes from a string. The juxtapositional phase is the same as in mGAs. A pseudo algorithm for fast messy GAs is shown in Figure 45.

```
1.    perform probabilistically complete initialization
      evaluate fitness of all population members
2.    for i = 1 to the maximum number of primordial generations
          perform tournament selection
          if (a building block filtering event is scheduled) then
              perform building block filtering
              evaluate fitness of all population members
          end if
      end loop
3.    for i = 1 to the maximum number of juxtapositional generations
          perform cut-and-splice
          perform other operators (currently not used)
          evaluate fitness of all population members
          perform tournament selection
      end loop
```

Figure 45. Pseudo Algorithm for Fast Messy GAs

B.5.2 Fast Messy Genetic Algorithm Parameters. Fast messy GAs need a building block filtering and thresholding schedule instead of the population size reduction schedule required by mGAs. Goldberg provides formulas for deriving schedules (43:60–61), but the formulas contain additional parameters and no guidance is given for choosing their values. The remainder of mGA parameters are used by fmGAs as well.

B.5.3 Mathematical Theory of How (Why) Fast Messy GAs Work. Fast messy GAs are governed by the Schema Theorem (Equation 18) just like mGAs. The difference relates to how the population of "good" building blocks is created for processing by the juxtapositional phase. Goldberg performs a detailed analysis to show that a much smaller initial population of long strings (PCI) can be manipulated (through BBF) to create a population of "good" building blocks just as effectively as PEI and the primordial phase of mGAs (43:60–61).

B.5.3.1 Complexity Analysis. Reducing the overall time complexity of the algorithm is the main reason for switching from mGAs to fmGAs. PCI and BBF result in a time complexity of $\mathcal{O}(l \log l)$ for initialization and the primordial phase combined (43:61). Thus, the design goal has been met—fmGAs exhibit better efficiency than mGAs ($\mathcal{O}(l \log l)$ vs $\mathcal{O}(l^k)$) and preserve their effectiveness. Space complexity for fmGAs remains unchanged from SGAs and mGAs ($\mathcal{O}(nl)$) and populations can be sized much smaller than mGAs. Again, the time and space complexities of the evaluation function usually dominate those of the control sequence.

Appendix C. Background on Parallel Computing

This appendix contains background material on parallel computing, most of which has been presented in previous AFIT theses, Merkle (77). Two distinct but interdependent aspects of parallel computing are presented. Section C.1 considers issues related to the design and implementation of parallel computer architectures. Section C.2 examines the design and implementation of algorithms which exploit application parallelism.

C.1 Parallel Architectures

The vast majority of computer architectures in common use are based on the organization proposed by von Neumann in the 1940s, in which a single memory area is used to store both instructions and data. Such architectures are referred to as von Neumann-based. Von Neumann-based parallel processing systems can be categorized as Multiple Instruction Multiple Data (MIMD), Single Instruction Multiple Data (SIMD), Multiple Instruction Single Data (MISD), or Single Instruction Single Data (SISD). A special case of the MIMD category is the Single Program Multiple Data (SPMD) paradigm. Other architectures exist, but their use is primarily limited to research. The majority of commercially available parallel architectures are either SIMD or MIMD.

During any given instruction cycle, all of the processors of a SIMD architecture execute the same instruction, using different data. In order for the instructions to be applicable to the data on all the processors, they must be more general and therefore less powerful. As a result, the individual processors have small instruction sets, making them relatively inexpensive, so that it is cost effective for SIMD architectures to include large numbers of processors. SIMD architectures with 64,000 processors are fairly common. The tradeoff is that for a fixed amount of memory, there is less memory per processor.

In contrast, the processors of a MIMD architecture act independently, and can take advantage of more powerful instructions. Each processor is more expensive, so that MIMD architectures are typically implemented with fewer processors than SIMD architectures. This means that each processor can be allocated more memory.

A single processor and its allocated memory are together called a *node*. The relative computational power of each node in a parallel architecture is often referred to as the *granularity* of the architecture. Most SIMD architectures are categorized as *fine grained* because they have a large number of nodes, each of which has a simple processor with a small amount of memory. In contrast, most MIMD architectures are categorized as *coarse grained* because they have a relatively small number of nodes, each with a powerful processor and significant memory.

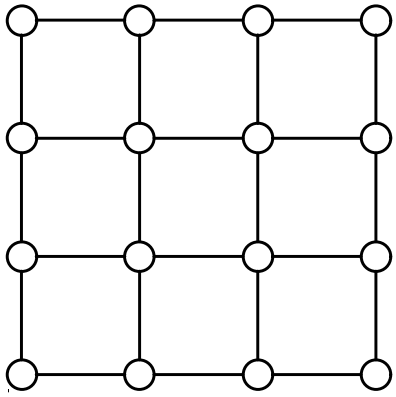


Figure 46. 4×4 Mesh Interconnection Network

Some architectures allow processors to access memory allocated to other processors, or simply allow all the processors to access a single global memory. Such architectures are referred to as *shared memory* architectures. Processors within such architectures can communicate data by storing it in memory which is accessible to the receiving processor. Most SIMD architectures are in this category. Most MIMD architectures, on the other hand, are *distributed memory*, meaning that processors cannot access memory allocated to other processors. These architectures are also referred to as *message passing*, because the processors communicate via communication links. This type of communication is generally very slow relative to other processor activities.

Parallel architectures can also be categorized according to their *interconnection topology*, or *network*, which defines the other processors to which each processor can communicate data. A common interconnection topology for SIMD architectures is a *2-D mesh*, in which the processors are arranged, logically if not physically, in a two dimensional array. A 4×4 mesh is shown in Figure 46. Mesh interconnection networks allow each processor to communicate data to each of the four processors at its sides. Well known examples of such systems are the Connection Machine, which is manufactured by Thinking Machines, Inc., and the Paragon, which is manufactured by Intel.

A common interconnection topology for MIMD architectures is a *hypercube*. Hypercube architectures have a *dimension*, N, and have 2^N processors. A hypercube of dimension 4 is shown in Figure 47. Each processor is directly connected to, and can communicate data to N other processors in a single step. Any processor can communicate data to any other processor in no more than N steps. One of many examples of a commercially available hypercube architecture is Intel Corporation's parallel supercomputer, the iPSC/i860.

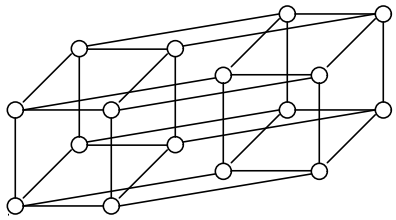


Figure 47. Dimension 4 Hypercube Interconnection Network

C.2 Parallel Algorithms.

Software development for parallel architectures is fundamentally different than for sequential architectures (9). The primary question in developing parallel software is whether to design parallel algorithms and implement them directly, or to implement sequential algorithms and then parallelize them. For most cases in which there is no sequential software predecessor, and even in some cases where there is such a predecessor, the first approach likely results in better performance.

Several algorithm properties can lead to performance improvements when the algorithms are implemented in parallel. The two most common such properties are data parallelism and control parallelism(73). The former describes a situation in which an algorithm processes multiple data items in the same way, and the actions taken for any particular data item do not depend on the results of processing other data items. The latter is present when two distinct operations on the same data do not depend on each other. Another form of parallelism, sometimes referred to as "trivial" parallelism, is present when two separate activities, which share neither control nor data, can be executed simultaneously.

Chandy and Misra propose an architecture independent method for the description of an algorithm(9). A UNITY (Unbounded Nondeterministic Iterative Transformations) program describes the requirements for a process. It does not specify the order of operations or the mapping of operations to processors. Thus, a UNITY program may be mapped to any architecture, whether it be sequential, asynchronous shared-memory, or distributed memory. The description of a mapping describes how the UNITY program is executed on the target architecture. Mappings for particular classes of architectures exhibit common characteristics. The target architecture in this study is a hypercube, which is a distributed memory (DM) system. Chandy and Misra describe DM systems formally as consisting of a fixed set of processors, a fixed set of communication channels, and a memory for each processor(9:83). As such, a mapping to such an architecture must

- allocate each statement in the program to a processor;
- allocate each variable to either a memory or a channel;
- specify the control flow for each processor;

- allocate at most one variable, which is of type sequence, to each communication channel;
- be such that a variable which is allocated to a channel is referenced in statements which are allocated to exactly two processors.

Furthermore, the statements allocated to one of the processors which reference a channel variable may only modify the variable by appending an item to the sequence. They may only do so when the sequence is of length less than a constant buffer size. Finally, the statements allocated to the other processor which references the channel variable may only modify the variable by removing the first item in the sequence. They may only do so when the length of the sequence is greater than zero.

Appendix D. PHGA Operation

Operation of PHGA is similar to the other AGCT binary GAs based on GENESIS. The only addition parameter is the number of nodes, which is entered at the command line. A sample input file is shown in Figure 48.

Traditionally, the GENESIS engine has expected an input file either named `in` or `in.something` where `something` is an unique file extension. If the run is to us the file named `in`, then no additional arguments are extended at the command line, i.e.

```
ga.energy
```

and no other arguments on the command line. If `in.something` is to be used as the input parameter file, then just the extension is entered following the executable name, i.e.

```
ga.energy something
```

where it must be the second item on the command line. Of course, standard UNIX command options can be included like & to run the process in the background and > to redirect standard output to a file or null output, `/dev/null`.

In a parallel environment the above option is unusable because additional parameters must be specified and the order varies from platform to platform. In response to these problems, two tokens, `psga_param` and `psga_default` were used. The first token `psga_param` indicates to the program that the next argument indicates the file to use for input. The second token `psga_default` indicates to the program use the GENESIS default input file, `in`.

For most MPI implementations, a script `mpirun` is used to hide machine dependent start-up procedures. A switch `-np` followed by an integer, indicates the number of nodes to used. An exception is the Intel Paragon internal version, available with Intel R1.04 OS. Here, no script is used, rather a switch `-sz` followed by an integer value indicates how many nodes to run. Examples follow:

```
mpirun -np 4 psga.energy psga_param 2.10.24
psga.energy -sz 4 psga.energy psga_param 2.10.24
```

the first is the mpirun version while the second is the Paragon internal version. Both are using four nodes and using input file `in.2.10.24`. With the Paragon version, standard UNIX command options can be included like `&` to run the process in the background and > to redirect standard output to a file or the bit bucket, `/dev/null`.

```
      Experiments = 1
     Total Trials = 500
  Population Size = 20
 Structure Length = 240
   Crossover Rate = 0.65
    Mutation Rate = 0.005
   Generation Gap = 1.0
   Scaling Window = 1
  Report Interval = 1
 Structures Saved = 1
Max Gens w/o Eval = 10
    Dump Interval = 0
      Dumps Saved = 0
          Options = ycel
  Number of Peaks = 1.0
Minimization Prob = 1.0
 Replacement Prob = 1.0
      Random Seed = 987654321
         Rank Min = 1.5
```

Figure 48. Sample input parameter file for PHGA

Appendix E. Genocop-III

This appendix provides details of the most recent implementation of Michalewicz's Genocop (GEnetic algorithm for Numerical Optimization of COnstrained Problems). GENOCOPIII is the new version of this system for handling Numerical Optimization of Problems with Linear and Non-Linear Constraints. It has been completely rewritten and incorporates no code from previous versions of the system. This version is written in ANSI-C and should compile on any system supporting the ANSI standard. The general algorithm is detailed in Section E.1. Input parameters are discussed in Section E.2. Genocop III has a much richer set of operators than "classical" GAs which are discussed in Section E.3

E.1 Algorithm

Genocop III combines the characteristics from all aspects of the Evolutionary Computation and other stochastic techniques. The general algorithm is presented in Figure 49. It features a unique repair method incorporated in the search population evaluation, Figure 50.

Procedure Genocop III
begin
  $t \leftarrow 0$, "t is number of generations"
  initialize $P_s(t)$
  initialize $P_r(t)$
  evaluate $P_s(t)$
  evaluate $P_r(t)$
  while (**not** termination-condition) **do**
  begin
    $t \leftarrow t + 1$
    select $P_s(t)$ from $P_s(t-1)$
    alter $P_s(t)$
    evaluate $P_s(t)$
    if $t \ mod \ k = 0$ **then**
    begin
      alter $P_r(t)$
      select $P_r(t)$ from $P_r(t-1)$
      evaluate $P_r(t)$
    end
  end
end

Figure 49. The structure of Genocop III

procedure evaluate $P_s(t)$
begin
  for each $\vec{s} \in P_s(t)$ **do**
    if $\vec{s} \in \mathcal{F}$ "feasibility set"
    then evaluate $\vec{s}$ (as $f(\vec{s})$) **else**
    begin
      select $\vec{r} \in P_r(t)$
      generate $\vec{z} \in \mathcal{F}$
      evaluate $\vec{s}$ (as $f(\vec{z})$)
      if $f(\vec{r}) > f(\vec{z})$ **then** replace $\vec{r}$ by $\vec{z}$ in P_r
      replace $\vec{s}$ by $\vec{z}$ in P_s with probability p_r
    end
end

Figure 50. Evaluation of population P_s in Genocop III

E.2 Input Parameters

The input parameters for Genocop-III can be divided into the following classes: Static, Domain Constraints [1] (DC), Linear Constraints (LC), and Operator Probability Distribution (OD). Additionally, we can say that the static constraints and operator probability distribution are artifacts of the *algorithm domain* while the domain and linear constraints are artifacts of the *application domain*. Since the objective is to effect the algorithm with respect to a specific application, only algorithm domain parameters will be study in this experiment set.

The static input parameters for Genocop-III are presented in table 25. The operator probability distribution is a sequence of real numbers indicating the relative frequency for each operator. If operator frequency control is *fixed*, these numbers are normalized by the algorithm. However, if operator frequency control is *adaptive*, operators with nonzero values are assigned a starting relative frequency midway between the upper and lower bounds defined in the header file `genocop.h`, while those with a zero value are assigned a relative frequency of zero.

Of the parameters listed in Table 25, several are essentially constant or experiment dependent. Theses are:

- Total number of variables
- Number of nonlinear equality constraints
- Number of nonlinear inequality constraints
- Number of linear inequality constraints
- Number of linear inequality constraints

[1] Here the phrase "Domain Constraints" is used in a context more limited than normally used in compute science, specifically, the allowable range of specified variables. If a domain constraint is not defined for a variable, it defaults to the architecture dependent range for R

Table 25. Static Input Parameters

Name	Description	Values
Number_Variables	Total number of variables	$N \cup 0$
Number_NLE	Number of nonlinear equality constraints	$N \cup 0$
Number_NLIE	Number of nonlinear inequality constraints	$N \cup 0$
Number_LC	Number of linear inequality constraints	$N \cup 0$
Number_DC	Number of variable constraints	$N \cup 0$
Ref_Pop_Size	Size of reference population	N
Search_Pop_Size	Size of search population	N
Number_Operators	Number of operators	N
Total_Evaluations	Number of total evaluations	N
Reference_Period	Period of evaluation of reference pop	N
Reference_Offspring	Number of offspring for each ref pop eval	N
Select_Reference_Pt	Selection of ref point to repair search point	0=random, 1=ordered
Repair_Method	Selection of repair method for search pop	0=random, 1=deterministic
Replace_Prob	Prob. of replacement for search pop	[0.0,1.0]
Reference_Init_Type	Init method for reference population	0=single, 1= multiple
Search_Init_Type	Init method for search population	0=single, 1= multiple
Object_Type	Objective function type	0=max, 1=min
Test_Case	Test case number	N
Epsilon	EPSILON for equalities	R
Seed1	Random number seed 1	$0 \leq iSeed1 \leq 31328$
Seed2	Random number seed 2	$0 \leq iSeed2 \leq 30081$
Frequency_Mode	Operator frequency control	0=fixed, 1=adaptive

- Number of variable constraints
- Objective function type
- Init method for reference population
- Init method for search population
- Test case number
- Random seeds (1 and 2)

In addition, the parameter EPSILON only applies to nonlinear equalities which are not at this time applicable to the REGAL approach, therefore in it is ignored. This leave the static parameters indicated in Table 10.

E.3 Operators

GENOCOP–III as available form Michalewicz, uses the 10 operators listed below. Each is explained in greater detail in the following sections.

1. Whole arithmetical crossover
2. Simple arithmetical crossover
3. Whole uniform mutation

4. Boundary mutation
5. Non-uniform mutation
6. Whole non-uniform mutation
7. Heuristic crossover
8. Gaussian mutation
9. Pool recombination operator
10. Scatter search operator

E.3.1 Whole arithmetical crossover. Arithmetic crossover, produces $\vec{z} = a\vec{x} + (1-a)\vec{y}$, from parents $\vec{x}$ and $\vec{y}$. Always produces a feasible solution (for $0 \leq a \leq 1$) in convex search spaces. Applied to all genes.

E.3.2 Simple arithmetical crossover. Same as above except crossover is only applied following a single point. Closest analogy in this implementation to *single point crossover* in GENESIS.

E.3.3 Whole uniform mutation. Single parent $\vec{x} = (x_1, ..., x_k, ...x_n)$ produces a single offspring $\vec{x}' = (x_1, ..., x_k', ...x_n)$ where x_k' is a random value (uniform probability distribution) from the range of element k. Discussion on this and the next two mutation operators is available in Michalewicz, Logan, and Swaminathan (80)

E.3.4 Boundary mutation. Same as whole uniform mutation except x_k' is either $left(k)$ or $right(k)$ with uniform probability. .

E.3.5 Non-uniform mutation. This unary operator is responsible for fine tuning of the system.

$$x_k^{t+1} = \begin{cases} x_k^t + \triangle(t, r(k) - x_k) & \text{if a random binary digit is 0} \\ x_k^t - \triangle(t, x_k - l(k)) & \text{if a random binary digit is 1} \end{cases} \tag{20}$$

for $k = 1, \ldots, n$. The function $\triangle(t, y)$ returns a value in the range $[0, y]$ such that the probability of $\triangle(t, y)$ being close to 0 increases as t increases (t is the evaluation number). This property causes this operator to search the space uniformly initially (when t is small), and very locally at later stages. In experiments reported by Michalewicz et al. (1994), the following function was used:

$$\triangle(t, y) = y \cdot r \cdot (1 - \frac{t}{T})^b, \tag{21}$$

where r is a random number from [0..1], T is the maximal evaluation number, and b is a system parameter determining the degree of non–uniformity.

E.3.6 Whole non-uniform mutation. Applies non-uniform mutation to the whole vector. A code review reveals that some components may not be mutated, while others may be mutated more than once.

E.3.7 Heuristic crossover. An interesting variation along this line is the *heuristic crossover* operator proposed by Wright (113); this crossover uses values of the objective function in determining the direction of the search, and it produces only one offspring. The operator generates a single offspring $\vec{z}$ from two parents, $\vec{x}$ and $\vec{y}$ according to the following rule:

$$\vec{z} = r \cdot (\vec{x} - \vec{y}) + \vec{x} \tag{22}$$

where r is a random number between 0 and 1, and the parent $\vec{x}$ is not worse than $\vec{y}$, i.e., $f(\vec{x}) \geq f(\vec{y})$ for maximization problems and $f(\vec{x}) \leq f(\vec{y})$ for minimization problems.

E.3.8 Gaussian mutation. The most popular mutation operator is *Gaussian mutation*, which modifies all components of the solution vector $\vec{x} = \langle x_1, \ldots, x_n \rangle$ by adding a random noise:

$$\vec{x}^{t+1} = \vec{x}^t + N(0, \vec{\sigma}) \tag{23}$$

where $N(0, \vec{\sigma})$ is a vector of independent random Gaussian numbers with a mean of zero and standard deviations $\vec{\sigma}$. Such a mutation is used in evolution strategies (Bäck et al., 1991) and evolutionary programming (Fogel, 1995). (One of the historical differences between these techniques lies in adjusting vector of standard deviations $\vec{\sigma}$).

E.3.9 Pool recombination operator. Picks bits from a random parent selected from a set and fills up the offspring vector. Returns index of the worst parent in the set. (from code comments)

E.3.10 Scatter search operator. The scatter search operator involves computing the centroid of group of parents and moving from the worst individual beyond the centroid point. More precisely, the operator selects $k > 2$ parents (set J), determines the best and the worst individual within the selected group ($\vec{b}$ and $\vec{w}$, respectively), computes the centroid $\vec{c}$ of the selected group with removed worst individual:

$$\vec{c} = \sum_{\vec{x_i} \in J - \vec{w}} \vec{x_i}/(k-1) \tag{24}$$

and computes the 'reflected point' $\vec{y}$ (i.e., the offspring) obtained from the worst one:

$$\vec{y} = \vec{c} + (\vec{c} - \vec{w}) \tag{25}$$

This operator embodies ideas originally presented, according to Michalewicz(81), by Glover in 1977 (38). The appeal of the scatter search in a GA is the concept an *orgy*, where the resulting offspring contains genetic material from more than two parents.

Appendix F. Statistical Methods

The chapter discusses statistical methods used in this research.

F.1 Analysis of Variance (ANOVA)

F.1.1 Single Factor Factorial Design . Suppose we have a treatments or different levels factors we wish to compare. Each of the n observed responses from each of the a treatments is a random variable. The observation can be described as a linear statistical model:

$$y_{ij} = \mu + \tau_i + \epsilon_{ij} \begin{cases} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, n \end{cases} \tag{26}$$

where y_{ij} is the (ij)th observation, μ is a parameter common to all treatments called the *overall mean*, τ_i is a parameter unique to the ith treatment called the *treatment effect*, and ϵ_{ij} is a random error component. For hypothesis testing, the model errors are assumed to be normally independently distributed random variables with mean zero and variance of σ^2.

Actually, the statistical model, Equation 26, describes two different situations with respect to the treatment effects. First the treatments could have been selected by the experimenter. This is a *fixed effects model*. The conclusions derived from hypothesis testing apply **only** to the treatment levels in the analysis. They can't be extended to treatments not explicitly considered. Alternatively, the treatments could be *random samples* from a larger population of treatments. This is a *random effects model*. The τ_i are random variables and knowledge about the particular ones investigated are relatively useless. Rather, the experimenter test hypothesis about the variability of τ_i and tries to estimate this variability.

Initially, consider an experiment where a single factor is tested at a levels. In addition, there are n replicates of each treatment levels. N is the total number of experiments and is equal to $\sum_{i=1}^{a} n_i$. The term *analysis of variance* is derived from a partitioning of total variability into its component parts. The total corrected sun of squares

$$SS_T = \sum_{i=1}^{a} \sum_{j=1}^{n} (y_{ij} - \bar{y}_{..})^2 \tag{27}$$

is used as a measure of overall variability in the data. Montgomery (87) shows how the total variability, as measured by SS_T can partitioned into a sum of squares of differences between the treatment averages and the grand average, plus a sum of squares difference of the difference of observations within treatments from the treatment average. The difference between the observed treatment averages and the grand average is a measure of the difference between the treatment means, whereas the difference of observations within a

treatment from the treatment average can be due to random error (or some other factor). Thus we have

$$SS_T = SS_{Treatments} + SS_E \tag{28}$$

where $SS_{Treatments}$ is called the sum of squares due to treatments and SS_E is the sum of squares due to error.

The formal test of the hypothesis of *no difference* in treatment means ($H_0 : \mu_1 = \mu_2 = \ldots = \mu_a$) or equivalently ($H_0 : \tau_1 = \tau_2 = \ldots = \tau_a = 0$). Montgomery shows via Cochran's Theorem (87:59) that if the null hypothesis of no difference in treatment means is true, the ratio

$$F_0 = \frac{SS_{Treatment}/(a-1)}{SS_E/(N-a)} = \frac{MS_{Treatments}}{MS_E} \tag{29}$$

is distributed as F with $a-1$ and $N-a$ degrees of freedom. Thus we reject H_0 if

$$F_0 > F_{\alpha,\ a-1,\ N-a} \tag{30}$$

where F_0 is computed from Equation 29. ANOVA test are usually summarized in table form, Table 26. To eliminate rounding error involved with averages, the sum of square terms are calculated as follows:

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{n} y_{ij}^2 - \frac{y_{ij}^2}{N} \tag{31}$$

$$SS_{Treatments} = \sum_{i=1}^{a} \frac{y_{i.}^2}{n} - \frac{y_{..}^2}{N} \tag{32}$$

$$SS_E = SS_T - SS_{Treatments} \tag{33}$$

The dot notation used with respect to the y variables is defined in Equations 34 and 35.

Table 26. Analysis of Variance Table for the Single-Factor, Fixed Effects Model

Source of Variation	Sum of Squares	DOF	Mean Square	F_0
Between Treatments	$SS_{Treatments}$	$a-1$	$MS_{Treatments}$	$F_0 = \frac{MS_{Treatments}}{MS_E}$
Error (within treatments)	SS_E	$N-a$	MS_E	
Total	SS_T	$N-1$		

$$y_{i.} = \sum_{j=1}^{n} y_{ij}, \qquad \bar{y_{i.}} = y_{i.}/n \qquad i = 1, 2, \ldots, a \tag{34}$$

$$y_{..} = \sum_{i=1}^{a} \sum_{j=1}^{n} y_{ij}\,, \qquad \bar{y_{..}} = y_{..}/N \tag{35}$$

F.1.2 Two Factor Factorial Design . The single factor analysis shown is the previous section is expanded to encompass multiple factors. The section details procedures for ANOVA of a two factor design. Here there are a levels of factor A and b levels of factor B. The linear statistical model for this design is

$$y_{ijk} = \mu + \tau_i + \beta_j + (\tau\beta)_{ij} + \epsilon ijk \begin{cases} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, b \\ k = 1, 2, \ldots, n \end{cases} \tag{36}$$

where μ is the overall mean effect, τ_i is the effect of the ith level of the first factor A, β_j is the effect of the jth level of the second factor B, $(\tau\beta)_{ij}$ is the interaction between τ_i and β_j, and ϵijk is the random error component. Hypothesis tests, are equality of A treatment levels

$$\begin{aligned} H_0 &: \quad \tau_1 = \tau_2 = \ldots = \tau_a = 0 \\ H_1 &: \quad \exists\ \tau_i \neq 0 \end{aligned} \tag{37}$$

, equality of B treatment levels

$$\begin{aligned} H_0 &: \quad \beta_1 = \beta_2 = \ldots = \beta_b = 0 \\ H_1 &: \quad \exists \beta_i \neq 0 \end{aligned} \tag{38}$$

, and A and B treatment interaction

$$\begin{aligned} H_0 &: \quad (\tau\beta)_{ij} = 0 \qquad \forall\ i,\ j \\ H_1 &: \quad \exists\ \tau_i \neq 0 \end{aligned} \tag{39}$$

The Table 27 shows the ANOVA for this model. For each test, the H_0 hypothesis

Table 27. Analysis of Variance Table for the Two-Factor, Fixed Effects Model

Source of Variation	Sum of Squares	DOF	Mean Square	F_0
A treatments	SS_A	$a-1$	$MS_A = \frac{SS_A}{a-1}$	$F_0 = \frac{MS_A}{MS_E}$
B treatments	SS_B	$b-1$	$MS_B = \frac{SS_B}{b-1}$	$F_0 = \frac{MS_B}{MS_E}$
Interaction	SS_{AB}	$(a-1)(b-1)$	$MS_{AB} = \frac{SS_AB}{(a-1)(b-1)}$	$F_0 = \frac{MS_{AB}}{MS_E}$
Error	SS_E	$ab(n-1)$	$MS_E = \frac{SS_E}{ab(n-1)}$	
Total	SS_T	$abn-1$		

The terms are computed as follows:

$$SS_T = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijk}^2 - \frac{y_{...}{}^2}{abn} \tag{40}$$

$$SS_A = \sum_{i=1}^{a} \frac{y_{i..}^2}{bn} - \frac{y_{...}^2}{abn} \tag{41}$$

$$SS_B = \sum_{j=1}^{a} \frac{y_{.j.}^2}{an} - \frac{y_{...}^2}{abn} \tag{42}$$

$$SS_{AB} = SS_{Subtotals} - SS_A - SS_B \tag{43}$$

$$SS_E = SS_T - SS_{Subtotals} \tag{44}$$

$$SS_{Subtotals} = \sum_{i=1}^{a}\sum_{j=1}^{b} \frac{y_{ij.}^2}{n} - \frac{y_{...}^2}{abn} \tag{45}$$

$$y_{i..} = \sum_{j=1}^{b}\sum_{k=1}^{n} y_{ijk}, \qquad \bar{y_{i..}} = y_{i..}/bn \quad i = 1, 2, \ldots, a \tag{46}$$

$$y_{.j.} = \sum_{i=1}^{a}\sum_{k=1}^{n} y_{ijk}, \qquad \bar{y_{.j.}} = y_{.j.}/an \quad j = 1, 2, \ldots, b \tag{47}$$

$$y_{ij.} = \sum_{k=1}^{n} y_{ijk}, \qquad \bar{y_{ij.}} = y_{ij.}/n \quad \begin{matrix} i = 1, 2, \ldots, a \\ j = 1, 2, \ldots, b \end{matrix} \tag{48}$$

$$y_{...} = \sum_{i=1}^{a}\sum_{j=1}^{b}\sum_{k=1}^{n} y_{ij}, \qquad \bar{y_{...}} = y_{...}/abn \tag{49}$$

F.2 Kruskal-Wallis H Test.

The Kruskal-Wallis H Test determines "whether or not the means from k independent samples are equal when the populations [cannot be assumed to be] normal"(1:544). The algorithm for the Kruskal-Wallis test is given at Figure 51, in which n is the total number of observations, k is the number of samples, and R_i is the rank of observation i within the population. The Kruskal-Wallis H Test is used throughout as a means of verifying results from ANOVA.

Suppose we have k independent samples from k populations. We wish to test the null hypothesis

H_0 : the samples are from identical populations
against the alternative hypothesis

H_1 : the populations are not identical
at the α level of significance.

1. Compute h. *Calculate*

$$h = \frac{12}{n(n+1)} \sum_{i=1} k \frac{R_i^2}{n_i} - 3(n+1)$$

2. Accept or reject H_0. *If $h > \chi^2_{k-1,\alpha}$, reject H_0; otherwise accept H_0.*

Figure 51. Kruskal-Wallis H Test Algorithm
(1)

Bibliography

1. Arnold O. Allen. *Probability, Statistics, and Queueing Theory: With Computer Science Applications.* Computer Science and Scientific Computing. Academic Press, Inc., San Diego, California, 1990.
2. I. P. Androulakis, C. D. Maranas, and C. A. Floudas. Prediction of oligopeptide conformations via deterministic global optimization. *Journal of Global Optimization*, 1996. In Press.
3. Thomas Bäck. *Evolutionary Algorithms in Theory and Practice.* Oxford Press, New York, 1996.
4. Giles Brassard and Paul Bratley. *Algorithmics: Theory and Practice.* Prentice Hall, Englewood Cliffs NJ, first edition, 1988.
5. Donald J. Brinkman. Genetic algorithms and their application to the protein folding problem. Ms thesis, afit/gce/eng/93d-02, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1993.
6. Bernard R. Brooks, Robert E. Bruccoleri, Barry D. Olafson, David J. States, S. Swaminathan, and Martin Karplus. Charmm: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4(2):187–217, 1983.
7. J. P. Cahoon, W. N. Martin, and D. S. Richards. A multi-population genetic algorithm for solving the k-partition problem on hyper-cubes. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 244–248, San Mateo, CA, July 1991. Morgan Kaufmann Publishers.
8. Hue Sun Chan and Ken A. Dill. The protein folding problem. *Physics Today*, pages 24–32, February 1993.
9. K. Mani Chandy and Jayadev Misra. *Parallel Program Design: A Foundation.* Addison-Wesley Publishing Company, Reading MA, 1989.
10. Thomas Coleman, David Shalloway, and Zhijun Wu. A parallel build-up algorithm for global energy minimizations of molecular clusters using effective energy simulated annealing. *Journal of Global Optimization*, 4:171–185, 1994.
11. Committee on Physical, Mathematical, and Engineering Sciences. *Grand Challenges 1993: High Performance Computing and Communications.* Office of Science and Technology Policy, 1992.
12. Committee on Physical, Mathematical, and Engineering Sciences, Federal Coordinating Council for Science, Engineering, and Technology, and Office of Science and Technology. High performance computing and communications—towards a national information infrastructure. Supplement to the President's Fiscal Year 1994 Budget, 1994.
13. Thomas E. Creighton, editor. *Protein Folding.* W. H. Freeman, New York, 1992.
14. Thomas E. Creighton. *Proteins: Structures and Molecular Properties.* W. H. Freeman, New York, 1993.
15. Lawrence Davis, editor. *Handbook of Genetic Algorithms.* Pitman, London, 1987.
16. Lawrence Davis. Adapting operator probabilities in genetic algorithms. In *International Conference on Genetic Algorithms*, pages 61–76, 1989.
17. Kenneth De Jong and William Spears. On the state of evolutionary computation. In Setphanie Forrest, editor, *Proceedings of the Fifth Interantional Conference on Genetic Algorithms*, pages 618–623, San Mateo, CA 94403, July 1993. Morgan Kaufmann Publishers, Inc.
18. Kenneth A. De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, 1975.
19. Kenneth A. De Jong. Adaptive system design: A genetic approach. *IEEE Transactions on Systems, Man and Cybernetics*, 10(9), September 1980.

20. Kenneth A. De Jong. On using genetic algoriths to search program spaces. In *Genetic Algorithms and their Applications: Proceedings of the Second Internation Conference on Genetic Algorithms*, pages 210–216, Hillsdale, NJ 07642, 1987. Lawrence Erlbaum Associates, Publishers.

21. Marco Dorigo and Vittorio Maniezzo. Parallel genetic algorithms: Introduction and overview of current research. *Parallel Genetic Algorithms*, pages 5–35, 1993.

22. Bruce S. Duncan. Parallel evolutionary programming. In *The Second Annual Conference on Evolutionary Programming*, pages 202–208, San Diego, CA 92121, 1993. Evolutionary Programming Society.

23. Andrew Dymek. Examination of hypercube implementations of genetic algorithms. Master's thesis, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1992.

24. Hesham El-Rewini, Theodore G. Lewis, and Hersham H. Ali. *Task Scheduling in Parallel and Distributed Systems.* Prentice Hall Series in Innovative Technology. Prentice Hall, Englewood Cliffs, NJ 07632, 1994.

25. Larry J. Eshelman, Richard A. Caruana, and J. David Schaffer. Biases in the crossover landscape. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 10–19, San Mateo, CA, June 1989. Morgan Kaufmann Publishers, Inc.

26. Larry J. Eshelman and J. David Schaffer. Preventing premature convergence in genetic algorithms by preventing incest. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 115–122, San Mateo, CA, 1991. Morgan Kaufman Publishers.

27. Steve Fairchild, Ruth Pachter, and Perrin Ronald. Protein structure analysis and prediction. *The Mathematica Journal*, 5(4):64–69, 95.

28. Terence C. Fogarty. Varying the probability of mutation in the genetic algorithm. In *International Conference on Genetic Algorithms*, pages 104–109, 1989.

29. David Fogel, editor. *Proceedings of the Second IEEE Conference on Evolutionary Computation*, Rundle Mall, South Australia, 1995. Causal Productions Pty Ltd. Published on CD-Rom.

30. David B. Fogel. On the philosophical differences between evolutionary algorithms and genetic algorithms. In *The Second Annual Conference on Evolutionary Programming*, pages 23–29, San Diego CA, 1993. Evolutionary Programming Society.

31. L. J. Fogel, A. J. Owens, and M. J. Walsh. *Artificial Intelligence Through Simulated Evolution.* Wiley Publishing, New York, 1966.

32. Lawrence J. Fogel. The future of evolutionary programming. *IEEE—ACSSC*, pages 1036–1038, 1990.

33. Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In *Foundations of Genetic Algorithms 2.* Morgan Kaufmann Publishers, Inc., 1993.

34. M.R. Garey and D.S. Johnson. *Computers and Intractibility: A Guide to the Theory of NP-Completeness.* Freeman, 1979.

35. George H. Gates, Jr. Predicting protein structure using parallel genetic algorithms. Ms thesis, afit/gcs/eng/94d-03, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1994.

36. Robert L. Gaulke. The application of hybridized genetic algorithms to the protein folding problem. Ms thesis, afit/gcs/eng/95d-03, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1995.

37. Jr. George H. Gates, Ruth Pachter, Laurence D. Merkle, and Gary B. Lamont. Simple genetic algorithm parameter selection for protein structure prediction. In Fogel (29). Published on CD-Rom.

38. F. Glover. Heuristics fir integer programming using surrogate constraints. *Decision Sciences*, 8(1):131–140, 1977.

39. David E. Goldberg. The theory of virtual alphabets.

40. David E. Goldberg. Optimal initial population size for binary-coded genetic algorithms. Technical Report TCGA Report Number 850001, The Clearing House for Genetic Algorithms, Department of Engineering Mechanics, University of Alabama, Alabama 35486, November 1985.

41. David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison–Wesley Publishing Company, Reading MA, 1989.

42. David E. Goldberg, Kalyanmoy Deb, and James H. Clark. *Genetic Algorithms, Noise, and the Sizing of Populations*, chapter 6, pages 333–362. Complex Systems Publications, Inc., 1992.

43. David E. Goldberg, Kalyanmoy Deb, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, San Mateo, CA, July 1993. Morgan Kaufmann Publishers.

44. David E. Goldberg, Kalyanmoy Deb, and Bradley Korb. Messy genetic algorithms revisited: Studies in mixed size and scale. In *Complex Systems*, pages 415–444, 1990.

45. David E. Goldberg, Kalyanmoy Deb, and Bradley Korb. Don't worry, be messy. In *International Conference on Genetic Algorithms*, pages 24–30, 1991.

46. David E. Goldberg, Bradley Korb, and Kalyanmoy Deb. Messy genetic algorithms: Motivation, analysis, and first results. In *Complex Systems*, pages 493–530, 1989.

47. David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 41–49, San Mateo CA, 1993. Morgan Kaufmann Publishers, Inc.

48. V. Scott Gordon and Darrell Whitley. Serial and parallel genetic algorithms as function optimizers. In Stephanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183, San Mateo, CA, July 1993. Morgan Kaufmann Publishers, Inc.

49. J. J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Transactions on Systems, Man & Cybernetics*, pages 122–128, 1986.

50. John J. Grefenstette. Learning by analogy in genetic classifier systems. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 291–297, San Mateo, CA, June 1989. Morgan Kaufmann Publishers, Inc.

51. John J. Grefenstette. A user's guide to genesis 5.0. Technical report, Vanderbuilt University, Nashville, TN, 1990.

52. John J. Grefenstette. Lamarckian learning in multi-agent environments. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 303–310, San Mateo, CA, July 1991. Morgan Kaufmann Publishers.

53. John J. Grefenstette. *Deception Considered Harmful*, pages 75–90. Foundations of Genetic Algorithms 2. Morgan Kaufmann, 1992.

54. William E. Hart, Scott Baden, Richard K. Below, and Scott Hohn. Analysis of the numerical effect of parallelism on a parallel genetic algorithm. In *Proceedings of International Parallel Processing Symposium*, pages 606–611. IEEE, 1996.

55. John H. Holland. *Adaptation in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor MI, 1975.

56. John H. Holland. Genetic algorithms. *Scientific American*, 267(1):66–72, July 1992.

57. Horton, Moran, Ochs, Rawn, and Scrimgeour. *Principles of Biochemistry.* Prentice Hall, 2nd edition, 1996.

58. R. Jaenicke. Protein folding: Local structures, domains, subunits, and assemblies. *Biochemistry*, 30:3147–3161, 1991.

59. Charles E. Kaiser, Jr. Refined genetic algorithms for polypeptide structure prediction–auxiliary volume. Ms thesis, afit/gce/eng/96d-13, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 96. In Process.

60. Charles E. Kaiser, Jr., Laurence D. Merkle, George H. Gates, Jr., Gary B. Lamont, and Ruth Pachter. Performance analysis of a parallel hybrid genetic algorithm using mpi. Presented at the 28th Central Regional Meeting of the American Chemical Society, Dayton, OH, June 1996.

61. Charles E. Kaiser, Jr., Laurence D. Merkle, George H. Gates, Jr., Gary B. Lamont, and Ruth Pachter. Exploiting domain knowledge in genetic algorithms for polypeptide structure prediction–a preview. Presented at the 28th Central Regional Meeting of the American Chemical Society, Dayton, OH, June 1996.

62. Charles E. Kaiser, Jr., Laurence D. Merkle, George H. Gates, Jr., Gary B. Lamont, and Ruth Pachter. Exploiting domain knowledge in genetic algorithms for polypeptide prediction. Presented at the 212th National Meeting of the American Chemical Society, Orlando Florida, August 1996.

63. Charles E. Kaiser, Jr., Laurence D. Merkle, George H. Gates, Jr., Gary B. Lamont, and Ruth Pachter. Real-valued and hybrid genetic algorithms for polypeptide structure prediction. In *Applied Computing 1997: Proceedings of the 1997 Symposium on Applied Computing*, New York, 1997. The Association for Computing Machinery. Accepted for Publication.

64. Stuart A. Kauffman. *The Origins of Order, Self-Organization and Selection in Evolution.* Oxford Unbiversity Press, New York, 93.

65. Lydia Kronsjö and Dean Shumsheruddin, editors. *Advances in Parallel Algorithms.* Halsted Press, New York, 1992.

66. Vipin Kumar, Ananth Grama, Anshul Gupta, and George Karypis. *Introduction to Parallel Computing: Design and Analysis of Algorithms.* The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

67. Gary B. Lamont, editor. *Compendium of Parallel Programs for the Intel iPSC Computers.* Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB OH 45433, 1993.

68. Roland E. Larson and Robert P. Hostetler. *Calculus with Analytic Geometry.* D.C. Heath and Company, Lexington, MA, 3rd edition, 1986.

69. Scott M. LeGrand and Kenneth M. Merz Jr. The application of the genetic algorithm to the minimization of potential energy functions. *Journal of Global Optimization*, pages 49–66, 1993.

70. Scott M. LeGrand and Kenneth M. Merz, Jr. Preface. In Kenneth M. Merz, Jr. and Scott M. LeGrand, editors, *The Protein Folding Problem and Tertiary Structure Prediction.* Birkhäuser, Boston, 1994.

71. Thomas Lengauer. Algorithmic research problems in molecular bioinformatics. *Arbeitspapiere der GMD 748*, May 1993.

72. Shem-Tov Levi and Ashok K Agrwala. *Real Time System Design.* McGraw-Hill Computer Science Series. McGraw-Hill Publishing Company, New York, 1990.

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

74. Zhenqin Li and Harold A. Scheraga. Monte carlo-minimization approach to the multiple-minima problem in protein folding. *Proceedings of the National Academy of Science USA*, 84:6611–6615, 1987.

75. Robert C. Martin, IV. A gain scheduling optimization method using genetic algorithms. Ms thesis, afit/gae/eng/94d-, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1994.

76. Herbert Meislich, Howard Nechamkin, and Jacob Sharefkin. *Theory and Problems of Organic Chemistry*. Schaum' Outline Series. McGraw-Hill, Inc, New York, 2 edition, 1991.

77. Laurence D. Merkle. Generalization and parallelization of messy genetic algorithms and communication in parallel genetic algorithms. Ms thesis, afit/gce/eng/92d-08, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1992.

78. Laurence D. Merkle, Robert L. Gaulke, George H. Gates, Jr., Gary B. Lamont, and Ruth Pachter. Hybrid genetic algorithms for polypeptide energy minimization. In *Applied Computing 1996: Proceedings of the 1996 Symposium on Applied Computing*, New York, 1996. The Association for Computing Machinery.

79. Z. Michalewicz. A hierarchy of evolution programs: An experimental study. *Evolutionary Computation*, 1(1):51–76, 1993.

80. Z. Michalewicz, T.D. Logan, and S. Swaminathan. Evolutionary operators for continuous convex parameter spaces. In A.V. Sebald and L.J. Fogel, editors, *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, pages 84–97, River Edge, NJ, 1994. World Scientific Publishing,.

81. Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation Journal*, September 1996. Based on to be reviewed version.

82. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 2nd edition, 1994.

83. Zbigniew Michalewicz, August 1996. Personal e-mail to author.

84. Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, New York, 3rd edition, 1996.

85. Zbigniew Michalewicz and Girish Nazhiyath. Genocop III: a co-evolutionary algorithm for numerical optimization. In Fogel (29), pages 647–651. Published on CD-Rom.

86. Thierry Montcalm, Weili Cui, Hong Zhao, Frank Guarnieri, and Stephen R. Wilson. Simulated annealing of met-enkephalin: low-energy states and their relevance to membrane-bound, solution and solid-state conformations. *Molecular Structure (Theochem)*, 308:37–51, 1994.

87. Douglas C. Montgomery. *Design and Analysis of Experiments*. John Wiley and Sons, New York, 3rd edition, 1991.

88. H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as fucntion optimizer. *Paralle Computing*, 17(7):619–632, 1991.

89. Akbar Nayeem et al. A comparative study of the simulated-annealing and monte carlo-with-minimization approaches to the minimum-energy structures of polypeptides: [met]-enkephalin. *Journal of Computational Chemistry*, 12(5):594–605, 1991.

90. Thomas J Ngo, Joe Marks, and Martin Karplus. Computational complexity, protein structure prediction, and the levinthal paradox. chapter 14, pages 433–506. 1994.

91. Office of Technology Assessment. Mapping our genes—the genome projects: How big, how fast? Technical Report No. OTA-BA-373, U. S. Congress, U. S. Government Printing Office, Washington, D.C., 1988.

92. James B. Olsan. Genetic algorithms applied to a mission routing problem. Ms thesis, afit/gce/eng/93-12, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, December 1993.

93. Ruth Pachter, T. M. Cooper, R. L. Crane, and W. W. Adams. Smart structures and materials. *SPIE Proceedings 1*, 1993.

94. Donald A. Parish. A genetic algorithm approach to automating satellite range scheduling. Ms thesis, afit/gor/ens/94m-10, Graduate School of Engineering, Air Force Institute of Technology (AETC), Wright-Patterson AFB OH, 1993.

95. Judea Pearl. *Heuristics*. Addison–Wesley Publishing Company, Reading MA, 1989.

96. Chrisila C. Pettey and Michael R. Leuze. A theoretical investigation of a parallel genetic algorithm. In *Proceedings of the Third International Conference on Genetic Algorithms*, pages 398–405, San Mateo, CA, June 1989. Morgan Kaufmann Publishers, Inc.

97. William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, second edition, 1992.

98. Alfred A. Rabow and Harold A. Scheraga. Improved genetic algorithm for the protein folding problem by use of a Cartesian combination operator. *Protein Science*, 5(9):1800–1815, September 1996.

99. J. David Schaffer, Richard A. Caruana, Larry J. Eshelman, and Rajarshi Das. A study of control parameters affecting online performance of genetic algorithms for function optimization. In *Third International Conference on Genetic Algorithms*, pages 51–60, 1989.

100. J. David Schaffer and Larry J. Eshelman. On crossover as an evolutionarily viable strategy. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 61–68, San Mateo, CA, July 1991. Morgan Kaufmann Publishers.

101. Marc Snir, Steve Otto, Steven Huss-Lederman, David Walker, and Jack Dongarra. *MPI: The Complete Reference*. The MIT Press, Cambridge Massachusetts, 1996.

102. Anne H. Soukhanov et al., editors. *WEBSTER'S II New Riverside University Dictionary*. The Riverside Publishing Company, Boston, MA, 1984.

103. William M. Spears and Kenneth A. De Jong. Using genetic algorithms for supervised concept learning. *IEEE-CH*, 29(15):335–341, July 1990.

104. Piet Spiessens and Bernard Manderick. A massively parallel genetic algorithm: Implementation and first results. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 279–285, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

105. M. Srinivas and Lalit M. Patnaik. Genetic algorithms: A survey. *COMPUTER*, 27(6):17–26, June 1994.

106. T. Starkweather, S. McDaneil, K. Mathias, D. Whitley, and C. Whitley. A comparison of genetic sequencing operators. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 69–76, San Mateo, CA, July 1991. Morgan Kaufmann Publishers.

107. Thomas Sterling, Paul Messina, and Paul H. Smith. Enabling technologies for peta(fl)ops computing. Technical Report CCSF-45, California Institute of Technology, July 1994.

108. Lubert Stryer. *Biochemistry*. W.H. Freeman, New York, 4rd edition, 1995.

109. Reiko Tanese. Parallel genetic algorithms for a hypercube. In John J. Grefenstette, editor, *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms*, pages 177–183, Hillsdale, NJ, July 1987. Lawrence Erlbaum Associates, Publishers.

110. Darrel Whitley. The *genitor* algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In *International Conference on Genetic Algorithms*, 1989.

111. Darrell Whitley, V. Scott Gordon, and Keith Mathias. Lamarckian evolution, the Baldwin effect and function optimization. In *PPSN III*, pages 6–15.

112. Darrell Whitley, Timothy Starkweather, and D'Ann Fuquay. Scheduling problems and traveling salesmen: The genetic edge recombination operator. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 133–140, San Mateo, CA, July 1991. Morgan Kaufmann Publishers.

113. Alden H. Wright. Genetic algorithms for real parameter optimization. In Gregory J. E. Rawlins, editor, *Foundation of Genetic Algotithms*, pages 205–218, San Mateo, CA, 1991. Morgan Kaufmann.

Vita

First Lieutenant Charles E. Kaiser, Jr. enlisted in the United States Air Force in October of 1986. He was first assigned to the 1912th Computer Systems Group, Langley AFB, VA, as a Contingency and Mobility Plans Programmer supporting the Tactical Air Command, and later, Air Combat Command battle staffs. He earned his bachelor's degree in Information Science from Christopher Newport College, Newport News, VA through the Bootstrap Program in 1992. He was commissioned through Officer's Training School in 1993. After completing Basic Communication-Computer Systems Officer Training he was assigned to the 4th Space Operations Squadron at Falcon AFB, CO. Here he directed the Milstar Operational Communications Planners Course, training communications planners supporting the National Command Authorities and unified commands, worldwide. Lt Kaiser left Space Command in 1995 to attend the AFIT. His projected follow-on assignment is to Air Force Global Weather Central at Offutt AFB, NE.

Permanent address: 608 W. Washington St.
New Carlisle, OH 45344