QA
76.623
.B76
2009

ADAPTIVE REPRESENTATIONS FOR IMPROVING EVOLVABILITY, PARAMETER TUNING, AND PARALLELIZATION OF GENE EXPRESSION PROGRAMMING

by

Nigel P. A. Browne

B. Sc. Ryerson University, 2005

A thesis

presented to Ryerson University

in partial fulfillment of the

requirements for the degree of

Master of Science

in the Program of

Computer Science

Toronto, Canada, 2009

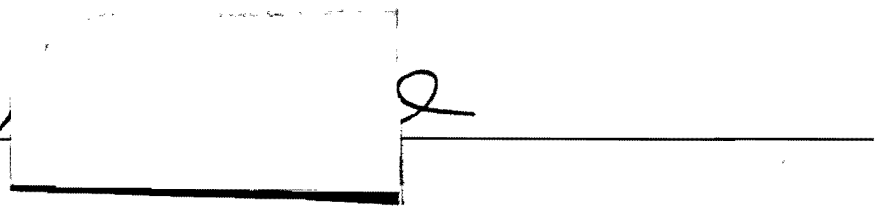
© Nigel P. A. Browne 2009

PROPERTY OF
RYERSON UNIVERSITY LIBRARY

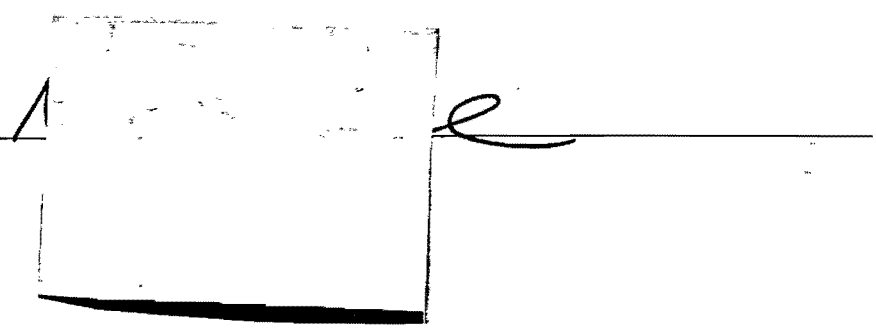
Author's Declaration

I hereby declare that I am the sole author of this thesis.

I authorize Ryerson University to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Signed: 

I further authorize Ryerson University to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signed: 

ADAPTIVE REPRESENTATIONS FOR IMPROVING EVOLVABILITY, PARAMETER TUNING, AND PARALLELIZATION OF GENE EXPRESSION PROGRAMMING

Nigel P. A. Browne

M. Sc. in Computer Science, 2009

Ryerson University, Toronto, Canada

Abstract

Gene Expression Programming (GEP) is a genetic algorithm that evolves linear chromosomes encoding nonlinear (tree-like) structures. In the original GEP algorithm, the genome size is problem specific and is determined through trial and error.

In this work, a novel method for adaptively tuning the genome size is presented. The approach introduces new mutation, transposition and recombination operators that enable a population of heterogeneously structured chromosomes, something the original GEP algorithm does not support. This permits crossbreeding between normally incompatible individuals, speciation within a population, increases the evolvability of the representations and enhances parallel GEP.

To test our approach an assortment of problems were used, including symbolic regression, classification and parameter optimization. Our experimental results show that our approach provides a solution for the problem of self-adaptively tuning the genome size of GEP's representation.

Acknowledgements

There are a number of individuals without whom this thesis would not have been possible and I would like to emphatically thank all of them for their assistance and input.

First and foremost, I need to thank my thesis supervisor Marcus dos Santos, for being particularly effective at directing my efforts, shaping my ideas, wading through the verbiage that was the early draft of my thesis, and knowing when to use the carrot and when to use the stick.

I'd like thank my friends and colleagues Dana Cotant and Greg Smith for their support throughout my master's degree.

My good friends (you know who you are) deserve a huge "thank you" for periodically pulling me out of my research bubble, making me laugh and generally reminding me that the world still existed.

Mom, Dad and my brother Drew, thanks for being my cheerleaders, occasionally feeding me, and listening to me ramble (incessantly) about my research.

Nigel P. A. Browne

Ryerson University

September 2009

Dedication

For Mom and Dad, thanks for the genes.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Evolutionary computation (EC) is a machine learning technique that uses processes often inspired by biological mechanisms to obtain a solution to a given problem. Applying an EC algorithm to a problem begins by defining how potential solutions are represented, which is known as the *problem representation*. A problem representation is defined by the type of input data (the *Terminal Set*) used to generate a solution, the desired number and types of outputs and the operations (the *Function Set*) used to transform the inputs into the output values . An important step in applying an EC methodology to a particular problem is the specification of parameters that define the problem representation and control the algorithm. Finding appropriate parameter values that yield satisfactory results usually requires carefully developed heuristics or expert knowledge. In EC algorithms, the concept of a *population* of candidate solutions, or individuals, is used to represent a pool of possible solutions to a particular problem. The encodings, or genomes, used to represent a solution vary depending on the EC methodology. It can be as simple as binary code, or as complex as a full fledged programming language. The Gene Expression Programming (GEP) algorithm [1], developed by Candida Ferreira, is an EC algorithm which uses separate encodings for the genotype and phenotype.

This thesis introduces novel enhancements to the Gene Expression Programming (GEP) algorithm that enable flexible genome representations, endow self adaptive characteristics, increases the diversity within a population and enhances the parallelization of the algorithm. The following issues are particularly relevant to the work presented here:

1. Evolvability: the structure of the problem representation does not vary during a run, as it is restricted to the initial values for the head domain length and number of genes. This constrains the algorithm to narrow bands of exploration and reduces its ability to produce

meaningful change or a paradigm shift within a population.

2. Crossbreeding and Speciation: in GEP, genetic operations and transformation are restricted to identically structured genomes, preventing different species, or disparately structured genomes, from evolving and competing within a population.

3. Distributed Evolution: parallelization is restricted by the inability for disparate populations to interact, slowing the exploration of the search space.

4. Parameter Tuning and Self Adaptation: the GEP algorithm lacks a self-adaptation mechanism and thus requires additional time and resources to systematically evaluate different control parameter sets and subjecting the algorithm to operator biases.

1.1 Approach

To address the evolvability of the problem representation, we developed two new operators to permit the structure of the GEP genome to be changed during a run. We call these new operators the *Adaptive Chromosome Size (ACS) Mutation* operator and the *Head Insertion Sequence* (HIS) Transposition operator.

The problems of speciation and genome interactions between disparately structured individuals was solved by replacing the canonical GEP recombination operators with modified versions that permit dissimilarly structured individuals to interact.

From the beginning of our explorations we wanted to improve the performance of the GEP algorithm when distributed. We quickly realized that transferring individuals between separate GEP populations was severely limited by the inability for structurally different individuals to recombine. This issue was eliminated by the introduction of our modified recombination operators.

Finally, to enable parameter tuning in the GEP algorithm, we designed our HIS and ACS mutation operators to eliminate the two critical parameters of the GEP algorithm, the head size and the number of genes. Additionally, the HIS and ACS mutation operators were designed to

permit the algorithm to self-adaptively tune the optimal chromosome structure.

Our proposed methodology was empirically evaluated using an assortment of problem classes and complexity levels. Symbolic regressions evaluated were kinematics problems, a series of polynomial regressions, and the "Sunspot Problem". The classification problem tested was the LiveDescribe dataset from the *The Center for Learning Technology* at *Ryerson University*. Finally, the effectiveness of the proposed methodology for optimizing parameters was evaluated using the De Jong test functions [2].

The effectiveness of the proposed changes were evaluated by comparing the performance of the enhanced GEP algorithm against the original GEP algorithm. Additionally, the symbolic regression results were compared to the adaptive distributed GEP algorithm developed by Park *et al.*. [3]. The results obtained using an application developed during the course of this thesis, known as *Syrah*, and the results were validated using the K-Fold method with 10 folds.

1.2 Contribution

The specific contributions of this work are:

1. Development of the Head Insertion Sequence (HIS) operator to self-adaptively tune the head size parameter in the GEP algorithm and to enable the structure of the individual to evolve during a run.

2. Creation of the Adaptive Chromosome Size (ACS) Mutation operator that self-adaptively tunes the number of genes of an individual in a GEP population. This further allows the genome structure to evolve.

3. Addition of new recombination operators to the GEP algorithm to enable structurally dissimilar genomes to interact. This also enables individuals to be transfered between separate GEP populations without any genomic structural constraints. This feature is particularly important to parallel GEP systems, as it permits unrestricted migration.

1.3 Overview of Thesis

The material following this introduction is organized as follows: Background material (Chapter 2), Materials and Methods (Chapter 3), presentation and discussion of our results (Chapter 4), and Conclusion and Future Work (Chapter 5).

Chapter 2 reviews prior work in evolutionary computation, specifically the GEP algorithm, distributed EC and various methods for parameter tuning. Additionally, we discuss material relevant to the development of our testing methodology and the *Syrah* system.

Chapter 3 introduces our new operators for the GEP algorithm, which solve the problems associated with the existing methods of parameter optimization in GEP. The new operators transform GEP populations from a collection of homogeneous individuals, with static sizes, to a dynamic population of heterogeneous individuals. This chapter also presents the *Syrah* system, which was developed to test and validate our hypothesis.

Chapter 4 presents the results of our experiments, which show that our new GEP operators perform better than previously explored methods. The new operators were tested using a variety of problems, and the performance was compared to other GEP-based methods.

Chapter 5 closes the thesis with the concluding remarks regarding the success of the new operators and a discussion of possible future work.

Chapter 2

Background and Related Work

This chapter presents the relevant existing research that pertains to the key issues addressed by this thesis, including: the canonical Gene Expression Programming algorithm, the evolvability of the problem representation, genome crossbreeding and speciation, distributed evolution, parameter tuning and self-adaptation.

2.1 Canonical GEP Algorithm

The Gene Expression Programming (GEP) algorithm was first published by Candida Ferreira in 2001 [1]. Like other EC methodologies, GEP derives its inspiration from biological processes and has been successfully applied to a variety of problems [4–9]. The outline of the canonical GEP algorithm is shown in Figure 2.1.

A significant difference in GEP, compared to Genetic Programming (GP) [10] or Genetic Algorithms (GA) [11], is the separation of the phenotype and genotype. Many existing methodologies (such as GP and GAs), use a single representation for both the genotype and the phenotype. By separating the representation, the GEP algorithm is able to benefit from the speed of operating on a linear genotype and the flexibility offered by the tree-based phenotype. It also permits the physical representation to affect the genetic code of the individual, as is found in nature.

In the GEP algorithm, each individual or candidate program is referred to as a chromosome. Every chromosome in the population represents a syntactically correct program, because of the underlying nature of the chromosome's encoding and representation.

PROPERTY OF
RYERSON UNIVERSITY LIBRARY

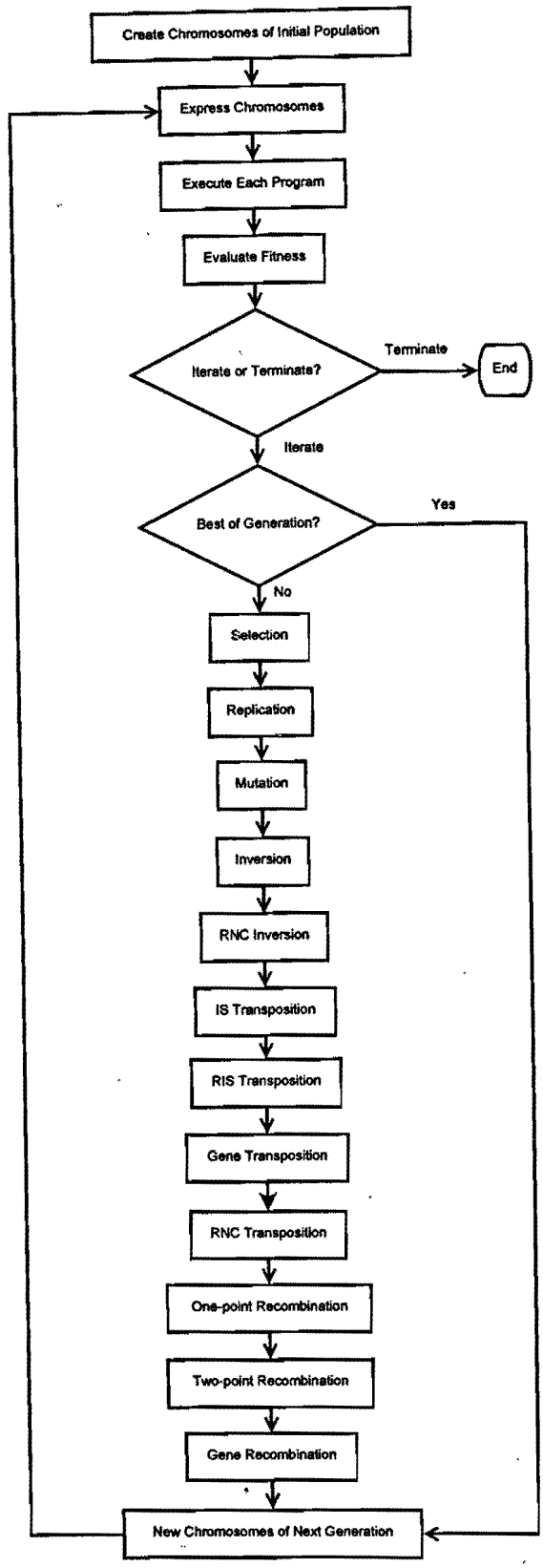


Figure 2.1: Flowchart of the canonical Gene Expression Programming algorithm [1].

2.1.1 Chromosome encoding

In GEP the genome (or *chromosome*) consists of a linear, symbolic string of one or more *genes*, each gene coding for an expression tree (ET). A gene consists of two adjacent regions called the *head*, containing symbols from both the function and terminal sets, and the *tail*, which encodes symbols from the terminal set. The tail only contains leaf nodes of the encoded ET, while the head may contain both leaf and internal nodes. In canonic GEP, both the number of genes and the head size of a gene are input parameters for the algorithm. The tail size t, in GEP, is a function of the head size h, and is determined as follows:

$$t = h(n_{max} - 1) + 1 \tag{2.1}$$

where n_{max} denotes the maximum arity found in the function set[1].

In the case of multigenic chromosomes, all ETs are connected by their root node using a *linking function*, which is a defined parameter. In the GEP system presented in this work we used the addition operator as the linking function. Figure 2.2 shows sample chromosome and the respective tree it encodes. The linear genome is encoded using Karva notation [1] and is translated into the expression tree by reading left to right and top to bottom. Using gene 1 in Figure 2.2 as an example, the first symbol (*) is used as the root node of the ET. Since the (*) operator has an arity of two, the next two symbols (a and /) are read from the linear genome and added as child nodes to the root of the ET. Next, since the (a) node is a terminal, it is a leaf node in the ET but the (/) node requires two child nodes (since it has an arity of two). The next two symbols are read from the linear genome (a and b) and added as child nodes to the (/) node. This completes the translation of the linear genome to the expression tree, since all of the leaf nodes consists of symbols from the terminal set. The two symbols remaining in the gene (a and a) are not used for the current translation, but could become active as a result of later changes to the genome. These regions that are not translated are called *introns*. Conversely, the region of the gene that is translated (designated by positions 0-4) is called the *Open Read Frame*. The

[1]Like in genetic programming, the function set is also a parameter to the GEP algorithm.

linking function, used to join the genes during evaluation, is not encoded in the chromosome because it is specified in the algorithm's parameters.

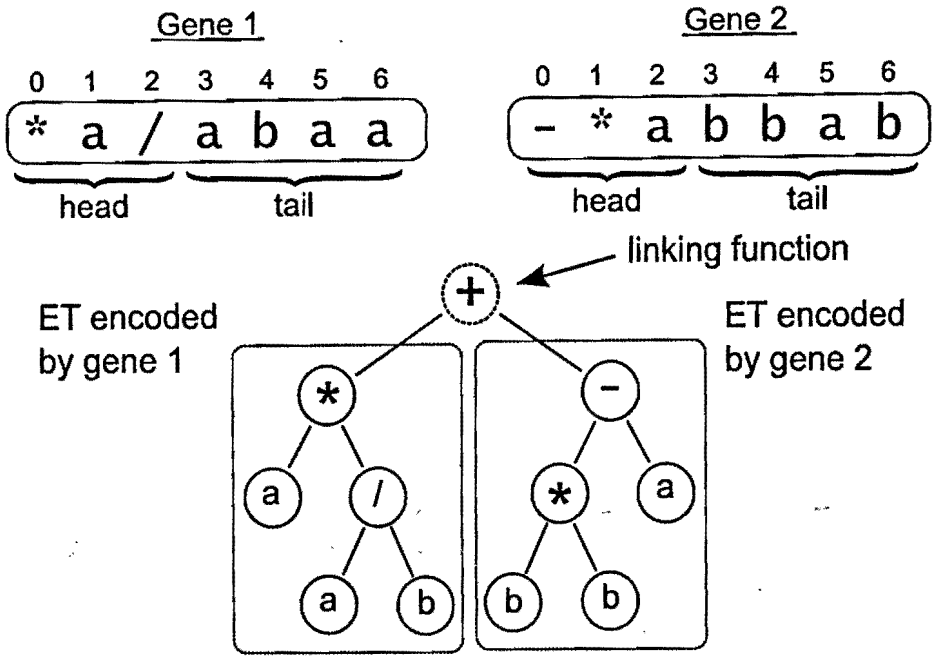


Figure 2.2: Chromosome with two genes, head size 3, tail size 4.

2.1.2 Standard Genetic Operators

The standard GEP algorithm implements several different classes of genetic operators, including: selection, mutation, inversion, transposition and recombination. Each operator promotes the exploration of the search space using different methods and have their own application rates.

Most genetic operators in GEP are applied in a different manner than they are applied in other methodologies, such as Genetic Programming. When an operator is applied to the population, a subset of the population determined by the operator rate (probability of application) is selected. Each individual in this subset then has the operator applied to it. This contrasts to applying the rate individually to each genome, as in genetic programming. The exception to this is the mutation operator, which is applied to each chromosome in the population.

The standard GEP mutation operator is the main source of genetic variation in the algorithm. It can function anywhere within the chromosome, using rules depending on where the selected point is located. For example, if the mutation is to occur in the head of a gene, then the mutated value may be any element from the function or terminal sets. However, if the mutation occurs in the tail only terminals may be used. As a result, the modified chromosome will always be syntactically correct. As mentioned earlier, the mutation operator is applied to every chromosome in the population. Since every chromosome undergoes mutation, the mutation rate refers to the number of point mutations in the genome.

The inversion operator is used to reverse the order of a section of a genome. Since parts of the genotype may not be translated into the phenotype, inversion allows non-coding regions to become active.

There are three methods for mixing the sequence of symbols (or codons) within a chromosome when using the GEP algorithm: Insertion Sequence (IS) Transposition, Root Insertion Sequence (RIS) Transposition and Gene Transposition. This family of operators selects a sequence and relocates it within the chromosome. The IS Transposition operator is permitted to select any sequence within the gene and insert it into any position except the root position of a gene. The RIS Transposition operator operates similarly to the IS Transposition operator, except it always inserts the selected sequence into the root position of the targeted gene. Since it is inserted into the root position, a sequence starting with a function is always selected. Finally, the Gene Transposition operator shuffles an entire gene within the chromosome.

The final class of standard genetic operators in the Gene Expression Programming algorithm are the Recombination, or Crossover, operators. The GEP algorithm supports three different recombination operators, each of which involves two chromosomes and creates two offspring. The One-Point recombination operator selects a single point along the chromosome pair and exchanges the genetic material after that point. The two point recombination operators selects two points within the chromosome and exchanges the codons between those points. Finally, the gene recombination operator swaps an entire gene between the two chromosomes.

2.2 Evolvability

Evolvability refers to the ability of a genome to change over time and to occasionally produce offspring that are more effective at a particular problem (and thus permit the algorithm to perform an effective search) [12, 13]. For evolutionary computation, this becomes significant for representations, such as GEP, that separate the phenotype from the genotype. In the case of this thesis, we focus on the evolvability of the structure and encoding of the genotype. This is particularly important in the case of GEP, since the canonical algorithm uses a fixed genome structure and the structure controlled by two problem-specific parameters.

Lopes and Weinert [14] proposed an enhanced GEP algorithm called *EGIPSYS* that varied the length of the head domain on a genome-level basis. The individuals, however, were composed of a fixed number of equal-length genes. This contrasts with the approach presented here, where each individual may have any number of genes and each gene may have a unique head length. Additionally, *EGIPSYS* did not implement the one-point recombination operator nor introduce operators to vary a chromosome's length. It also restricted the operation of the gene recombination operator to like-sized individuals. All of these issues are resolved in the method presented here.

In an attempt to improve the evolvability of the individuals in GEP, Yue *et al.* [15] proposed a crossover strategy *Valid Crossover Strategy* which would crossover all individuals in a population and create the subsequent population from the n-best valid chromosomes. This approach helped the evolution of the solution, but not the evolution of the genome structure itself.

Several different strategies for improving the GEP algorithm were presented by Tang *et al.* in [16]. A feature of interest that they developed was an adaptive mutation mechanism, which was essentially a fitness proportional mutation rate. On an individual basis, the mutation rate applied to a chromosome was inversely proportional to its fitness. Thus, highly fit individuals would have a lower mutation rate applied to them, reducing the number of potentially disruptive changes to chromosome. Conversely, poorly fit individuals were more likely to have significant mutation performed on their chromosomes. The implementation of the Adaptive Chromosome

Size Mutation Operator introduced in this work uses the idea of a fitness proportional mutation rate to preferentially mutate the number of genes in poorly fit individuals.

In this work we introduce new operators to improve the evolvability of GEP genomes. The new operators are the HIS transposition and ACS mutation operators, which allow the structure of a GEP genotype to change over time. The evolution of the genotype occurs in parallel to the exploration of the search space for a particular problem, but these two processes are fundamentally linked. These evolutionary processes are interconnected because changes in the genotype can permit the exploration of search space regions that may be inaccessible to other genome structures.

2.3 Crossbreeding and Speciation

The concept of crossbreeding and speciation embraced in this thesis is that of interactions between disparately structured, but fundamentally compatible, genomes. The idea of crossbreeding specifically refers to the ability for any individual, regardless of structure to reproduce and create viable offspring. The ability to crossbreed any individual permits a more genetically diverse population and enable unrestricted exploration of the search space by the algorithm.

Speciation, on the other hand, can have several different interpretations. In particular, it can refer to the ability for "sub-populations" to exist within a single main population for the purpose of "niching" [17]. Speciation and niching is used to promote diversity within a population, prevent (or limit) convergence and to address multi-modal problems where different areas of the solution space require different individuals [13]. Two methods for using speciation, or niching, are *Crowding* [2] and *Function Sharing* [18].

The *EGIPSYS* algorithm [14] permits different sized chromosomes within a population, which other systems, such as canonical GEP, AdaGep [19] and PGEP-O [3], do not support. All individuals in an EGIPSYS population, however, were required to have the same number of genes. This contrasts with our proposed methodology, which supports (and, in fact, encourages) populations consisting of individuals that have both differing head domain lengths and gene

counts.

Park *et al.* introduced a parallel system, PGEP-O [3], which attempted to dynamically tune specific parameters of the GEP algorithm. In the work, the individuals were constrained by the genome restrictions of canonical GEP, that is, only identically structured individuals were able to interact and exist withing a single population or island. This methodology was limited because the transfer of individuals between islands, or migration, could only occur between islands with identical gene counts and head domain sizes. The methodology presented in this thesis eliminates these constraints by creating new operators that do not restrict the interaction of genomes with fundamentally different structures.

The contributions presented in this thesis enable crossbreeding between disparately structured individuals in a GEP population, a feature unavailable in canonical GEP. This enables evolution of different species within a population, and while specifically implementing niching is beyond the scope of this work, it could be examined in the future.

2.4 Distributed Evolution

The intrinsic parallel nature of Evolutionary Computation (EC) can often be further exploited by distributing a given EC algorithm. Parallelization techniques can generally be classified by their granularity, defined as either fine grained or coarse grained models. Fine grained techniques commonly have low computational requirements, but higher communication needs and are well suited for multi-processor systems. Coarse gained models, on the other hand, tend to be computationally intensive but have lower communication requirements and are better suited to discrete computational nodes. The *Island Model* is a coarse grained technique that was popularized by [20] and has been shown to be fault tolerant [21]. The distributed system implemented to validate our methodology uses the island model.

The exchange of genetic material between islands, or *demes*, is referred to as migration. The structure of the connections between islands, or the topology, is bounded by the cases of isolated islands (no migration) and fully-connected islands (migration to all other demes) [22],

although dynamic topologies have been suggested [23]. In addition to the topology, the rate of migration, number of migrants and the migration policy control the flow of individuals between islands [24]. One aspect of a migration policy is whether the migration occurs synchronously (migrations occur at specific intervals with specific partners) or asynchronously (migrations occurred whenever a deme has a migrant to exchange) [25]. Interested readers are directed to [23, 24, 26–30] for more detailed information regarding migration.

The PGEP-O system [3] is an example of a parallel GEP algorithm which used two island groups. The first island group was a standard island model implementation, in which a single population of individuals was evolved on each island. The second island group used the first group's island as their "individuals" in an attempt to use a Genetic Algorithm (GA) to optimize the parameter settings of the island populations. Since the two island groups were needed, PGEP-O could only operate in a distributed mode. Additionally, since Park *et al.* [3] did not address the interaction of differing genome structures, migration between the islands could only occur between like-structured populations. This limited the algorithm's ability to explore the search space.

Lin *et al.* proposed a fine grained parallel GEP system [31] which exploited niching to improve the performance of the GEP algorithm. Based on their reported algorithm and data, they used a shared pool of like-structured individuals and empirically determined the GEP algorithm's parameter values.

A multi-objective parallel GEP system, *PGEP-AP* [32], also used the island model with migration. In addition to the standard migration mechanism PGEP-AP used a separate elitist population to store the best individuals from the various sub populations.

The PED-GEP algorithm introduced in [33] used a measure of diversity to guide evolution among parallel clients, however, the details of their parallelization lacked further specifics.

Du *et al.*, in [34], demonstrated a parallel GEP implementation that used Estimation of Distribution to improve the performance of the GEP algorithm. This system used asynchronous migration with a fully connected island model, that is, each island (population) could potentially interact with any other island.

Our approach used a fully-connected coarse-grained model with random migration and to removed the restrictions placed on the migration mechanism by canonical GEP's inability to support dissimilarly structured chromosomes in a single population. By permitting unrestrained migration, populations in a parallel setting are now able to freely exchange candidate solutions to enhance the solution quality and diversity.

2.5 Parameter Tuning and Self-Adaptation

Most Evolutionary Computation algorithms require a set of control parameters, which influence the process evolution to be configured based on the particular problem being explored. The process of setting these parameters often require complex heuristics, "rules of thumb" or specific knowledge from a domain expert. Thus, it is desirable to automatically tune the parameter values prior to executing the algorithm or to self-adaptively tune the parameters during the run.

In problem solving and optimization, the impossibility theory of "No Free Lunch" [35] has been postulated and roughly states that without *a priori* knowledge of a problem (to tailor the methodology to it) no single problem solving method is inherently better for all problem classes [36]. This has implications for any evolutionary algorithm and parameter tuning method, especially those that attempt to optimize the parameters prior to executing an evolutionary run and then use static values throughout the run [37]. Additionally, it has been shown [38] that optimal parameter values can vary throughout a single run. This implies that, while it may be impossible to determine optimal values for all problems and situations, it should be possible to evolve values that are "good enough". Additionally, it implies that methodologies that are able to optimize their parameter values dynamically have an inherent advantage over those that do not.

The PGEP-O system presented in [3] approached the issue of parameter tuning as a separate optimization problem that ran in parallel to the main evolutionary algorithm. This system was a parallel GEP implementation that used the island model to evolve solutions to the target

problem and a Genetic Algorithm (GA) running on a separate client to optimize the two GEP parameters. The head size and gene count parameters were optimized by using trial values on each GEP island and then reporting back to the GA parameter optimizer. This approach, while successful, suffered from several issues that are remedied by our proposed methodology. The PGEP-O algorithm required additional resources, since the parameter optimization was a separate calculation. Additionally, the GA optimizer had to wait for an entire run to complete before it was able to execute a new generation, which is problematic for long running evolutions.

The *DM-GEP* algorithm [33] introduced a dynamic mutation rate operator in an attempt to guide evolution. *DM-GEP* divided the execution of a run into three stages, the initial stage, the metaphase stage and the anaphase stage. Each stage was then assigned a specific mutation rate and the mutation rate used in each generation was progressively scaled, by a fixed amount, from one value to the next. In this manner, the number of generations executed in a run was directly related to the mutation rates. This approach did not, strictly speaking, tune the mutation parameter and was not self-adaptive, but did dynamically alter the rate and showed improvement over the standard GEP implementation.

Bautu *et al.* introduced in [19] an algorithm, called AdaGEP, for automatically tuning the number of genes of a GEP representation. The approach involved adding to the genome a bit array that maps each bit to a gene in the chromosome. The bit in each position of the array indicates if that gene would be included in the translation to an expression tree during the fitness evaluation. Specific genetic operators were designed to operate on this bit array, thus evolving an optimal mask. The AdaGEP algorithm was limited by the fact that the total number of genes in any chromosome could never change. Thus, there was little benefit to using that method versus using automatically defined functions, or homeotic genes, in GEP's jargon, to evolve the execution order of the genes. Additionally, the size of individuals in the algorithm's population could never change, so that even if fewer genes were required, the genetic operators would still be performed on the full chromosome.

The work presented in [15] included a method to vary the mutation and crossover rates during a run, based on the *Cloud Model* [39]. This methodology improved the performance of

the GEP algorithm, but was only applied to like-structured genomes.

Eiben *et al.* stated in their "Parameter Control in Evolutionary Algorithms" survey [37] that determining successful values for algorithm parameters in EC is a "grand challenge" problem.

The approach to parameter tuning and self adaptation presented in this thesis was accomplished using multiple techniques which work together to self-adaptively tune GEP parameters. To tune the head domain length and number of genes, we developed the HIS Transposition and ACS mutation operators. In addition to these operators, we created new recombination operators which allowed structurally disparate (and normally incompatible) genomes to be able to crossbreed and create viable offspring, which permits individuals with different head domain length and gene count parameters to compete within a single population.

Chapter 3

Methodology and Implementation

This chapter introduces the proposed enhancements to the GEP algorithm to address the issues identified in Chapter 1, to wit: the evolvability of the problem representation, genome speciation and crossbreeding, distributed evolution, and parameter tuning and self-adaptation in the canonical GEP algorithm. The chapter will introduce our proposed enhancements, the details of the implementation of the framework used for evaluation, and the experiments used to validate our hypothesis.

3.1 Proposed GEP Algorithm Enhancements

To address the issues of evolvability, crossbreeding, distributed evolution and parameter tuning found in canonical GEP, our proposed modifications to GEP include several new operators and also modifications to the existing recombination operators. The new operators introduced in the following section offer solutions to the problems of evolvability and the tuning of two critical parameters in GEP. The modified recombination operators were developed to permit speciation within a GEP population and to enhance distributed GEP populations. The operators are shown, with heavy borders and a grey background, in the context of the GEP algorithm in Figure 3.1.

3.1.1 Enhancements for Parameter Tuning and Evolvability

The original version of the GEP algorithm required that two critical parameters, the length of the head domain and the number of genes in the chromosome, be set to fixed values prior to the execution of a run. These parameters are generally domain and problem specific, which further exacerbates the problem of finding "good" values (not even particularly optimal ones) for the

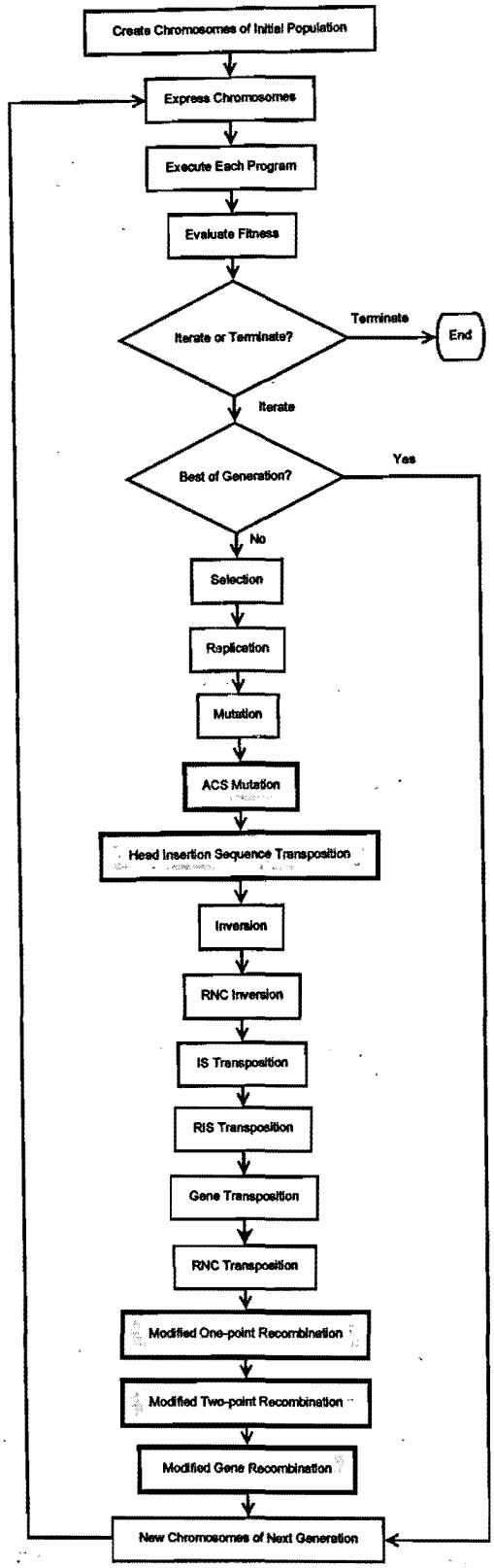


Figure 3.1: Flowchart of the proposed changes to the Gene Expression Programming algorithm.

parameters. By developing new operators which permit genome structure changes, we enabled the head domain length and number of genes to be implicitly tuned during a run. Our algorithm enhancements also permit each gene in a chromosome to have a unique head domain length. This extra feature enables the length of the gene to vary, and thus, the length of the function encoded by that gene.

In addition to parameter tuning, our approach improves the evolvability, or the ability of the structure of the genome to evolve, by removing the fixed length chromosome restrictions in canonical GEP and allowing the number of genes to vary during a run. Chromosome evolvability was specifically addressed by designing our new operators to increase the capacity of the genome for extracting and exploiting the underlying structure of the fitness function under consideration.

3.1.1.1 Adaptive Chromosome Size Mutation Operator

In Algorithm 1 we present the pseudo-code for the Adaptive Chromosome Size (ACS) mutation operator used in our enhanced GEP algorithm. The ACS operator mutates the number of genes in a chromosome, potentially increasing or decreasing the total number of genes when it is applied. The ACS operator is applied to the entire population during each generation.

The *AcsGeneMutation(...)* method takes a chromosome (*chr*) as a parameter and mutates it according to the following procedure. Initially, it calculates the *decayRate*, which is used to decrease the operator's application as the run progresses. In the *decayRate* calculation the *factor* is a user defined value that scales the *decayRate* and is set to 0.2 for all experiments. This scales the *decayRate* to zero for the final 20 percent of the run.

Next, the algorithm calculates the probability of mutation *muP*. The probability of mutation is inversely proportional to the individual's fitness when compared to the best fitness in the current generation. If the *muP* is less than the user defined minimum mutation rate, *minRate*, then *muP* is set equal to *minRate*. The mutation rate, *muP*, is then scaled by *decayRate* to arrive at the final *muP* value. The operator then generates a random probability using *RandProbability()* and compares it to *muP* to determine if the AcsGeneMutation will be applied to

Algorithm 1: ACS Mutation Operator Pseudocode

```
Data: Chromosome
Result: Mutated chromosome
begin
    /* Calculate the decay rate                                          */
    scaleFactor = 0.2
    decayRate = 1 - (gen + maxGen * scaleFactor) / maxGen
    /* Calculate the mutation rate, inverse to the fitness               */
    muP = (1 - chr.Ftn / bestFtn)
    /* Adjust the mutation rate if it is below the minimum               */
    if muP less than minRate then
        muP = minRate
    end
    /* Apply the decay to the mutation rate                              */
    muP = muP * decayRate
    /* Determine if mutation will occur                                  */
    if RandProbability() less than or equal to muP then
        /* Randomly decide to grow or shrink                             */
        if DoCoinToss() then
            /* Grow the chromosome by adding a new gene                  */
            insertionPoint=GetRnd(0, chr.NGenes)
            InsertGeneAt(insertionPoint)
        else
            /* Shrink the chromosome by deleting a gene, but only if we
               have atleast two genes                                    */
            if chr.NGene greater than 1 then
                deletionGene = GetRnd(1, chr.NGenes)
                DeleteGeneAt(deletionGene)
            end
        end
    end
end
```

the chromosome. Next, the operator performs a coin toss using *DoCoinToss()* to determine if a gene should be added or removed. When a gene is added, the operator selects an insertion point, *insertionPoint*, at a random position in the sequence of genes of the chromosome.

It then calls the worker method, *InsertGeneAt(...)*, to insert a randomly created gene at the insertion point. When a gene is removed, the operator first verifies that there is more than one gene (*chr.NGenes*) in the chromosome. It then randomly selects a gene in the chromosome using the *GetRnd(...)* method and calls the *DeleteGeneAt(...)* method to remove the gene from the chromosome.

The mutation operator always uses a step size equal to one. Thus, it modifies a single gene in the chromosome during each application of the operator. Alternative step sizes were not investigated, but will be examined in future work.

3.1.1.2 HIS Transposition Operator

To dynamically tune the size of a gene, we introduced a new transposition operator called *head insertion sequence transposition*, HIS transposition, for short. The transposable elements (also called *transposons*) in this case are fragments of the genome, located in the head of a gene, that can be activated and jump to (possibly) another gene head in the chromosome. Two features make this operator different from the canonic transposition operators used in GEP, to wit:

- the transposable element is necessarily located in the head of a gene; and
- during transposition, the transposon is cut from the place of origin (instead of copied, like in canonic transposition in GEP), thus *shortening* the length of the respective gene, and then inserted in the place of destination located necessarily in the head of (possibly) another gene, thus *elongating* the gene length at the target site.

Specifically, the HIS transposition operator works as follows. Initially the operator randomly chooses the chromosome, the start and end sites of the transposon, and the the target site. As mentioned above, these start and end sites are located in the head of a gene. Moreover,

transposons contain at most three elements. Next, the operator cuts the transposon from the site of origin, making the necessary arrangements to maintain the structural integrity of the gene. That is, if the transposon locates in the middle of the head of a gene, then the left and right remaining segments of the head are concatenated, thus forming the new gene head. Next, the operator inserts the transposon at the target site, thus elongating the head of the gene. Notice that the gene heads' at the place of origin and at the target site have now changed, the latter is now longer by say, k elements, and the former is k elements shorter. Finally, using Equation (2.1), the operator adjusts the respective new tail sizes of those genes. If the tail requires extra material, it is cut from the remaining genetic material in the source gene's tail.

3.1.2 Speciation and Crossbreeding

The notion of *species* is not present in canonic GEP, as all chromosomes have the same structure, *i.e.*, all individuals in a population have the same gene head size, same gene tail size and the same number of genes. The possibility of different species within a single GEP population is highly desirable feature for the parallelization of the algorithm, particularly when using a migration mechanism in a distributed setting. By modifying the existing GEP recombination operators to handle genomes with different structures, our enhanced GEP algorithm now supports crossbreeding and speciation within both a single population and distributed islands.

3.1.2.1 Recombination Operators for Nonuniform Chromosomes

To support different sized chromosomes created by ACS mutation and HIS transposition operations, we created modified versions for the one point and two points recombination operators used in GEP. These operators also facilitate integrating individuals with differing genome structures (*i.e.*, a differing number of genes and head domain lengths) into a target population during migration, when distributed. Recombination via these operators works as follows: initially the first positions for the head and tail sections of the two parent chromosomes are paired (see Figure 3.2). Then the crossover point (or points, in the case of two points recombination) is

randomly chosen from the overlapping sections of the chromosome. The crossover point locates either in the head of a gene or in the tail. If it falls in the *head*, then the genetic material is exchanged (the strands swapped) at the crossover point (see Figure 3.2(a)). For this case, there is no need to adjust the structure (tail size) of the gene containing the crossover point. If it locates in the *tail* of a gene, then we use the following process to exchange the genetic material of the genes where the crossover point is located. First we exchange the genetic material at the point of crossover. Then, we verify the tail sizes of the resulting genes comply with the respective resulting head sizes. If the tail size of a recombined gene is s elements shorter than the allowed size, then we append to it s elements from the tail of the other parent gene, thus making the final tail size of the recombined gene compliant with its head size (notice the strand added to O_1 in Figure 3.2(b)). On the other hand, if the tail size of the recombined gene is s symbols longer than the allowed size, then we cut its s last symbols out (notice the strand removed from O_2 in Figure 3.2(b)).

The rest of genetic material is exchanged as in normal crossover, with a caveat: for the case of GEP-RNC (GEP with real number constants [7]), if the crossover point locates in the tail of a gene, the genetic material in the domain of constants (Dc) is exchanged as normal and the length of the Dc domains are adjusted. If the crossover point falls in the Dc domain, then recombination proceeds via the same procedure used for the tails, as illustrated in Figure 3.2(b). The arrays containing the gene's real number constants are exchanged in their entirety [40].

Analogous to GEP, our recombination operators also produce two children from the parents. When the recombination point falls within the head region, one child having the same length as one of the parents, and the other child having the same length as the other parent. However, when the recombination point falls in the tail, the tail regions length of the children may need to be modified (using material from the parent), to satisfy Equation 2.1.

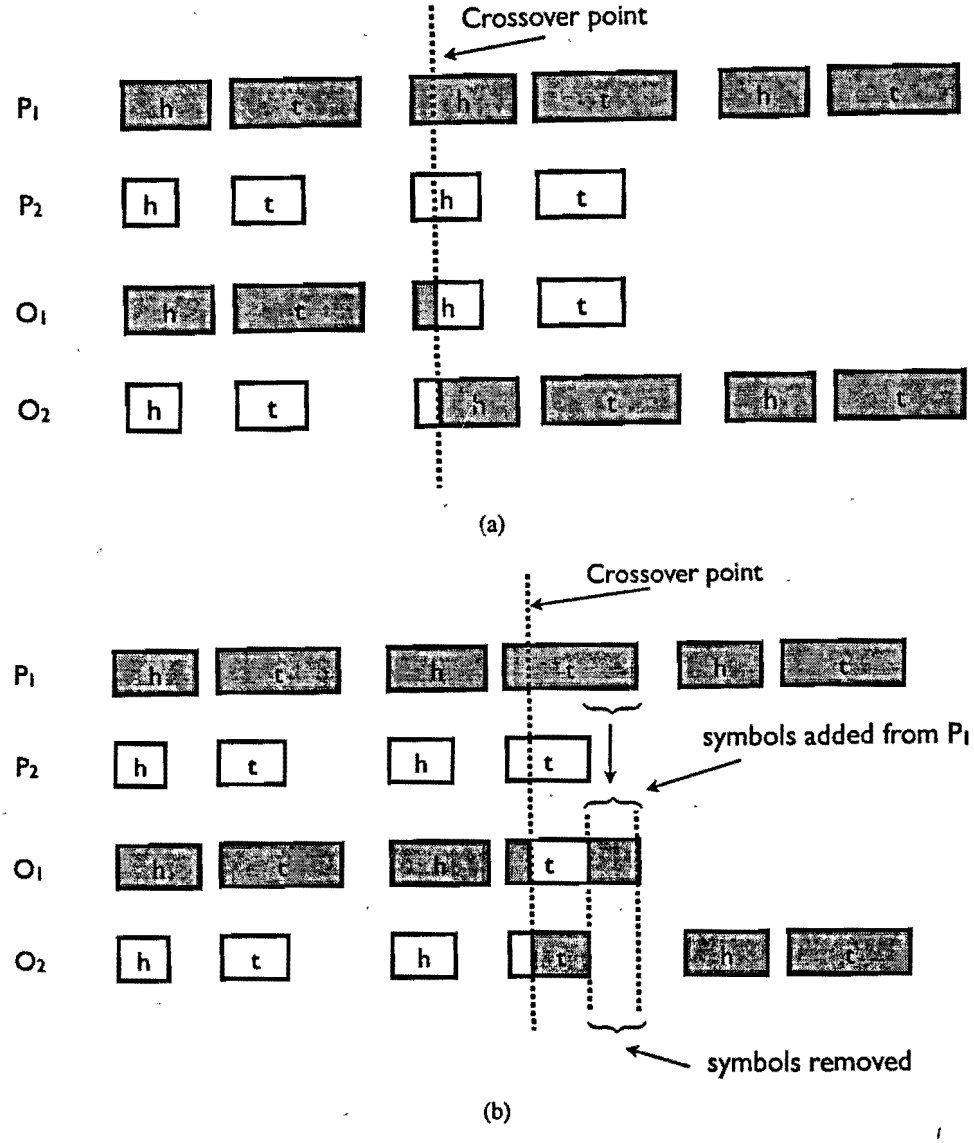


Figure 3.2: One-point recombination of two chromosomes, P_1 and P_2, containing 3 and 2 genes, respectively; *h* and *t* denote the head and tail portions of each gene, respectively. In Figure (*a*) the crossover point locates in the head of a gene. In Figure (*b*) the crossover point locates in the tail of a gene.

3.2 *Syrah* Implementation

In this study, a parallel capable GEP system called *Syrah*, which dynamically tunes the number of genes and gene size was developed. To test this system, a suite of non-trivial symbolic regressions was used and the quality of the models was benchmarked against models obtained via a canonic GEP system and competing methodologies.

Syrah's system requirements differ from GEP-RNC (GEP with real number constants [7]) in regards to the genetic operators it uses, which are detailed in Sections 3.1.1.1, 3.1.1.2, and 3.1.2.1.

In *Syrah*'s implementation, tournament selection with elitism was used. Tournament selection involves randomly selecting two individuals from the population, comparing their fitness values and then adding the more fit individual to the population used for the next generation. When elitism is used, the best individual from each generation is carried over to the next generation. Many GEP implementations use Roulette Wheel selection, but as long as elitism is used, various selection methods will produce equally good results [7].

When the *Syrah* system is operating in parallel, it uses a coarse grained technique (or Island Model [20]) to distribute the populations. *Syrah* uses the proposed genetic operators to permit disparate genome structures to be integrated into a given population during a migration event.

3.2.1 Development and Runtime Environments

All components of the research system *Syrah* were written in C# using the Microsoft .Net Framework version 3.5 and developed using Microsoft Visual Studio 2008. Data storage and management was accomplished using Microsoft SQL Server 2005 running on *Microsoft Windows XP Professional*. The client computers also used the *Microsoft Windows XP Professional* operating system.

3.2.2 Parallelization

Different methods and techniques exist for operating an EC algorithm in parallel. Generally parallel techniques can be divided into two categories, fine grained and coarse grained [24]. Fine grained techniques involve parallelizing the evaluation of the test cases and usually have more intensive communication requirements. Alternatively, coarse grained techniques distribute populations and have lower communication requirements, but higher computational needs. Our experimental system uses a common coarse grained technique known as the Island Model [20] to distribute populations to discrete computational nodes.

The network communication between nodes was implemented using the HTTP v1.1 protocol over an SSL connection. The server node is designed to listen for client requests on port 443, the standard port used by SSL web servers. Additionally, the communication between the client and the server is always initiated by the client. This combination of techniques was selected so that the communication would be relatively secure and to facilitate communication between the client and server, when the client was located behind a firewall. This circumvented firewall issues in the original network used for testing.

3.2.2.1 The Island Model

The Island Model [20] is a coarse grained method for parallelizing an EC algorithm that involves distributing a population (or subset of a population) to discrete computation nodes. Each computational node, or client, is responsible for independently executing a full evolutionary run and only reports it's final results to the managing server node.

Each node also has the ability to exchange individuals with other islands in the topology. This exchange of genetic material is referred to as migration and helps maintain diversity amongst the islands. The Island Model implemented in *Syrah* is a fully-connected topology that supports random-random migrations, meaning that a migration event can (randomly) involve any node in the system. Details regarding migration can be found in [22–24, 26, 27, 29, 41].

Finally, based on [21], the nodes do not implement any special handling for detecting and

preventing network topology faults. When a client is unable to complete a run (*i.e.*, because the host was restarted, the network failed, *etc*,), the client is simply starts a new run when it rejoins the *Syrah* topology.

3.2.3 Population Initialization

With the use of our recombination operators, the population is able to support individuals with different chromosome sizes. To take advantage of this feature, the population is seeded with randomly sized chromosomes. Both the number of genes and the head domain length of each gene are varied during this phase. The number of genes in each individual is randomly selected from between 1 and 10. During the creation of the chromosome, each gene selects a random head domain length between 5 and 15. These values were empirically determined during initial testing and were found to provide good genetic diversity. Additionally, we selected the random initialization method over a "ramped half and half" method [10] as a result of early experimentation.

The elements of the head are selected from a weighted bag. If the function set is smaller than that of the terminals, then the probability of selecting a function is $1/2$, otherwise they are equally weighted.

3.3 Experimental Design

An assortment of problems, of varying types and difficulty, were selected to evaluate the performance of our approach. The problems were selected from three areas to which Evolutionary Computation is commonly applied:

1. Symbolic regression, or the automatic synthesis of functions.
2. Classification, or generating boolean results (or labels) from a set of input values.
3. Parameter Optimization, or the automatic discovery of parameter values which produce a maximum and/or minimum for a given function.

Each experiment was performed using k-fold validation with 10 folds and 30 runs per fold. Each experiment consisted of two sets: a baseline set and an adaptive set. The baseline runs were executed using the standard GEP-RNC algorithm implemented as a part of the *Syrah* system with parameter tuning disabled. The adaptive runs were then executed in the same manner, but using the methodologies outlined previously.

Each experiment was executed using the *Syrah* framework's parallel mode, which uses the island model to distribute the populations to separate computational nodes. The experiments used 32 islands that were executed on 16 dual-core Intel computers, running the Windows XP Professional operating system. The *Syrah* system supports migration between the islands, but to facilitate the statistical analysis of the results, these experiments were run without this feature.

The baseline experiments were performed repeatedly using the values presented in Tables 3.1, 3.2 and 3.3. During the adaptive evolution runs, the number of genes and the size of the head domain were tuned using our new operators. The details of the initial chromosome lengths can be found in Section 3.2.3.

3.3.1 Symbolic Regression Experiments

The first three problems selected were the same problems used by H.H. Park *et al.*. in [3]. These were selected so that the performance of this methodology could be compared to an existing (parallel) GEP-based self-adaptive approach. The fourth experiment was a regression of a sawtooth wave, while the fifth experiment was a more difficult time series analysis problem.

The common algorithm configuration parameters are outlined in Table 3.1 and were shared amongst all of the experimental setups. The baseline experiments all produced poor results for gene counts of 1 through 3, which required 900 (3 x 10 folds x 30 runs per fold) runs to evaluate.

Table 3.1: Common Symbolic Regression Run Parameters

Selection Method	Elitist Tournament
Parameter	Value
Number of Generations	100
Population Size	50
Initial Head Size	5-15
Initial Number of Genes	1-10
One point recombination rate	0.5
Two point recombination rate	0.1
Gene recombination rate	0.1
Mutation rate	0.07
Minimum ACS Mutation Rate	0.05
IS transposition rate	0.1
RIS transposition rate	0.1
HIS transposition rate	0.1
Gene transposition rate	0.1
Function set	+, -, * , /
Linking function	+
K-Fold Validation	10 folds
Evolutionary Clients (*Syrah*)	31

3.3.1.1 Experiment 1

The first problem evaluated was a kinematics symbolic regression that modeled the movement of a vertically fired object. The kinematic equation for the position of the object at time t is defined by the following equation:

$$S(t) = S_0 + V_0 t + \frac{at^2}{2} \tag{3.1}$$

If we use an initial velocity, $V_0 = 25m/s$, an initial position of $S_0 = 0$, and assume the acceleration is equal to earth's gravity, $a = -9.8m/s^2$, then we can simplify the equation as:

$$S(t) = 25t + \frac{-9.8t^2}{2} = 25t - 4.9t^2 \tag{3.2}$$

For this experiment, fifty data points were sampled from the interval $t = 0.1$ to $t = 5$ and used as the test cases.

3.3.1.2 Experiment 2

Our second experiment extended the first, using two independent variables instead of one. Modifying equation 3.1 with the same assumptions as in experiment one, but with an independent initial velocity, gives:

$$S(t) = vt - 4.9t^2 \tag{3.3}$$

The test cases for this experiment were generated using V_0 values of 20, 25 and 30. The values of t were the same as in the first experiment.

3.3.1.3 Experiment 3

The third symbolic regression experiment used a fourth order polynomial that was used in [3] and similar to the ones used in [1,7].

$$y = -2.5x^4 + 4.6x^3 + 3x^2 + 2x + 1 \tag{3.4}$$

The algorithm attempted to evolve the function from 10 equally spaced samples taken from values of the Polynomial (3.4), in the interval $x = [1,10]$.

3.3.1.4 Experiment 4: Sawtooth Wave

The fourth experiment was a regression of a sawtooth wave, which has been used as a benchmark in other works [42]. The function is defined by:

$$F(x) = \sum_{i=0}^{n} \left(\frac{1}{i} \sin(i\,x) \right) : n = 1,...,9 \tag{3.5}$$

The dataset consisted of 250 equally spaced data points in the range $x = [-8,8]$. This range was selected instead of the 40 points in $[-1,1]$ used in [42] after discovering that the algorithm required a more challenging set of inputs.

3.3.1.5 Experiment 5: Wolfer Sunspot Time Series Prediction

The final experiment attempted to create a predictive model using 100 observations from the well known Wolfer Sunspot Series [43]. The data was formatted for time series analysis, using a delay time of 1 and an embedding dimension of 10. This dataset has also been used to evaluate other GEP systems, including [7] and [14].

3.3.2 Classification Experiments

Classification is a common and important task that evolutionary computation algorithms are applied to. The classification experiment performed in this work used a large, real world classification problem from the from the *The Center for Learning Technology* (CLT) at *Ryerson University.*

The evaluation of the classification experiments was accomplished using the "Hits with Penalty" method, as described in [7].

Table 3.2 lists the algorithm configuration values that were used for the classification experiments.

3.3.2.1 The LiveDescribe Dataset

The LiveDescribe project [44] is a software application developed by the *Center for Learning Technology* (CLT) at *Ryerson University* to add video descriptions (for the deaf) to video content. The project had originally used a manual process to select regions of dialog verses non-dialog in video content, so that descriptive video captions could be programmatically added to the non-dialog sections. Since the process of selecting the non-dialog regions was a manual and user intensive process, the CLT modified their application using a human designed classifier system. This system was, on average, 70% effective.

The dataset consists of six real value inputs and a single boolean output per tuple. Part of what makes this dataset a challenge is it's size. The initial dataset consisted of approximately 90,000 tuples. The input variables are the audio metrics RMS standard deviation, RMS average,

Table 3.2: Classification Experiment Run Parameters

Selection Method	Elitist Tournament
Parameter	Value
Number of Generations	175
Population Size	75
Initial Head Size	5-15
Initial Number of Genes	1-10
One point recombination rate	0.5
Two point recombination rate	0.1
Gene recombination rate	0.1
Mutation rate	0.07
Minimum ACS Mutation Rate	0.05
IS transposition rate	0.1
RIS transposition rate	0.1
HIS transposition rate	0.1
Gene transposition rate	0.1
Function set	+, -, * , /, sqrt, exp, sin, cos, tan, floor, ceiling, OR, AND, $<$, $>$, $\leq$, $\geq$, ==, !=
Linking function	+
K-Fold Validation	10 folds
Evolutionary Clients (*Syrah*)	31

a measure of audio entropy, zero crossing above to below, zero crossing left skew and a zero crossing low energy measurement. These inputs were sampled once for every 1 second of audio.

3.3.3 Parameter Optimization Experiments

The five parameter optimization test functions were selected from the the well known De Jong test functions [2]. These test functions were originally selected by De Jong to test the effectiveness of a given EC algorithm over a broad class of problems. While attempts have been made to improve the test set, it remains the *de facto* standard for parameter optimization validation. The five functions are presented here in their original form, but were modified (where necessary) to change them all to maximization functions, which allows for simpler evaluation with the GEP algorithm.

Table 3.3 lists the algorithm configuration values that were shared amongst all of the parameter optimization experiments.

Table 3.3: Common Parameter Optimization Run Parameters

Selection Method	Elitist Tournament
Parameter	Value
Number of Generations	100
Population Size	50
Initial Head Size	1-15
Initial Number of Genes	1-10
One point recombination rate	0.5
Two point recombination rate	0.1
Gene recombination rate	0.1
Mutation rate	0.07
Minimum ACS Mutation Rate	0.05
IS transposition rate	0.1
RIS transposition rate	0.1
HIS transposition rate	0.1
Gene transposition rate	0.1
Function set	+, -, * , /
Linking function	+
K-Fold Validation	10 folds
Evolutionary Clients (*Syrah*)	31

3.3.3.1 De Jong F1: Sphere Model

The first function in the De Jong test set is a three dimensional parabola that is convex, unimodal and continuous. The function has a maximum of 78.6 at $(x_1,x_2,x_3) = (\pm 5.12, \pm 5.12, \pm 5.12)$.

$$f(x_1,x_2,x_3) = \sum_{i=1}^{3} x_i^2 : -5.12 \leq x \leq 5.12 \tag{3.6}$$

3.3.3.2 De Jong F2: Rosenbrock's Function

The second function in the De Jong test set was first proposed by Rosenbrock [45] and is commonly referenced in optimization literature. This function is non-convex, unimodal and continuous, with a maximum of 3905.93 at $(x_1,x_2) = (-2.048, -2.048)$.

$$f(x) = 100 \times (x_1^2 - x_2)^2 + (1 - x_1)^2 : -2.048 \leq x \leq 2.048 \tag{3.7}$$

3.3.3.3 De Jong F3: Step Function

The third De Jong test function is a five dimension step function that is discontinuous, non-convex, unimodal and piece-wise constant. De Jong originally selected this function to test the ability for algorithms to handle discontinuities [2]. This function is restricted to $-5.12 \leq x \leq 5.12$ for testing. This function has a known maximum of 25 when the inputs are held at 5.12.

$$f(x) = \sum_{i=1}^{5} x_i : -5.12 \leq x \leq 5.12 \tag{3.8}$$

3.3.3.4 De Jong F4: Quadratic Function with Noise

The fourth test function in the De Jong collection is a noisy quadratic function that is continuous, unimodal, convex and has a high dimensionality. The function uses a Gaussian function to add noise. The function was limited to $-1.28 \leq x \leq 1.28$. This experiment used alternative values for the number of generations and the population size than the other parameter optimization experiments. This experiment had 350 generations and 500 individuals in the population.

The maximum of this function is approximately 1248.2 and occurs when all inputs are equal to ± 1.28.

$$f(x) = \sum_{i=1}^{30} i \times x_i^4 + Gauss(0,1) : -1.28 \leq x \leq 1.28 \tag{3.9}$$

3.3.3.5 De Jong F5: Shekel's Foxholes

This is a two dimension function that is continuous, non-quadratic and non-convex, with 25 local maximums. It was originally suggested by Shekel [46]. This version [47] of the function has maximum of approximately 499.002.

$$f(x,y) = 500 - \frac{1}{0.002 + \sum_{j=0}^{24} 1/[1 + i + (x - a(i))^6 + (y - b(i))^6]} \tag{3.10}$$

where

$$a(i) = 16 \times (i \bmod 5 - 2) \tag{3.11}$$

$$b(i) = 16 \times \left(\left\lfloor \frac{i}{5} \right\rfloor - 2 \right) \tag{3.12}$$

$$-65.523 \leq x \leq 65.523$$

Chapter 4

Results and Discussions

This chapter presents the results of the experiments outlined in Chapter 3 that were used to validate our enhancements to the GEP algorithm and that address the issues identified in Chapter 1.

4.1 Symbolic Regression Results

Table 4.1 shows a summary of the experiment results, including the best individual's fitness and chromosome size[1]. The best fitness is expressed as a percentage of the number of fitness cases solved. The visualized results and performance of the experiments are shown by Figures 4.1-4.10.

4.1.1 Discussion of Symbolic Regression Experiments

There are two figures for each of the first four experiments performed. The first figure of each pair shows the minimum, maximum and average chromosome lengths in the population

[1]Note that the size of a chromosome (*i.e.*, the length of the chromosome string) depends on its number of genes and the head size of each gene.

Table 4.1: Summary of symbolic regression experimental results

Exper. Number	Ours		Comparison	
	Length	Fitness	Length	Fitness
1[1]	254	99.984 %	266	99.496 %
2[1]	87	99.983 %	282	99.907 %
3[1]	155	99.735 %	470	96.187 %
4[2]	62	99.987 %	185	99.966 %
5[2]	55	99.179 %	186	98.936 %

1: Compared to PGEP-O

2: Compared to canonical distributed GEP

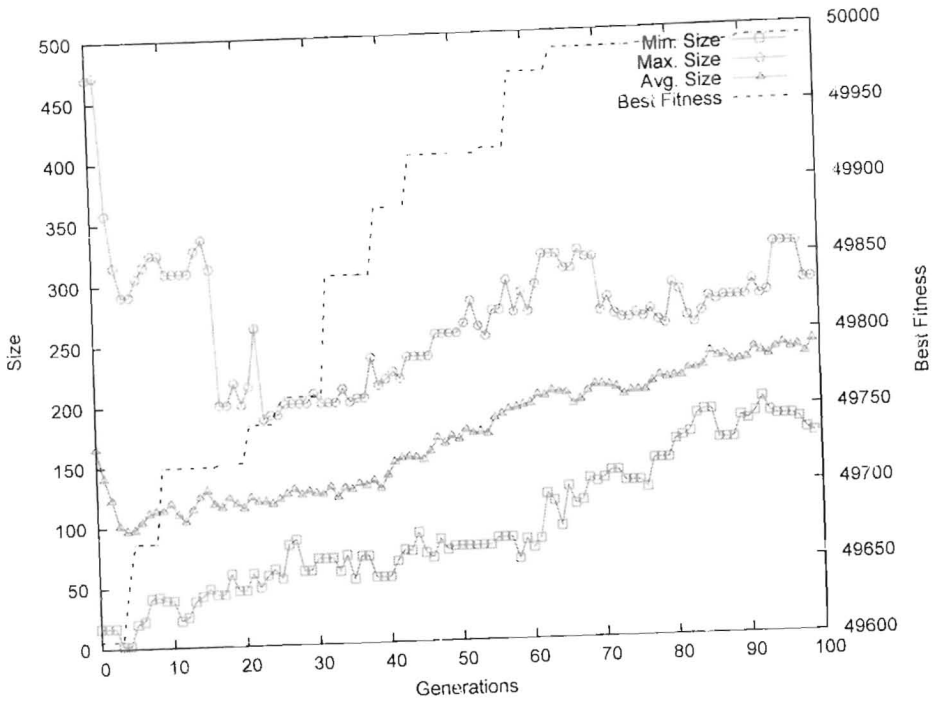


Figure 4.1: Symbolic regression experiment 1: chromosome sizes

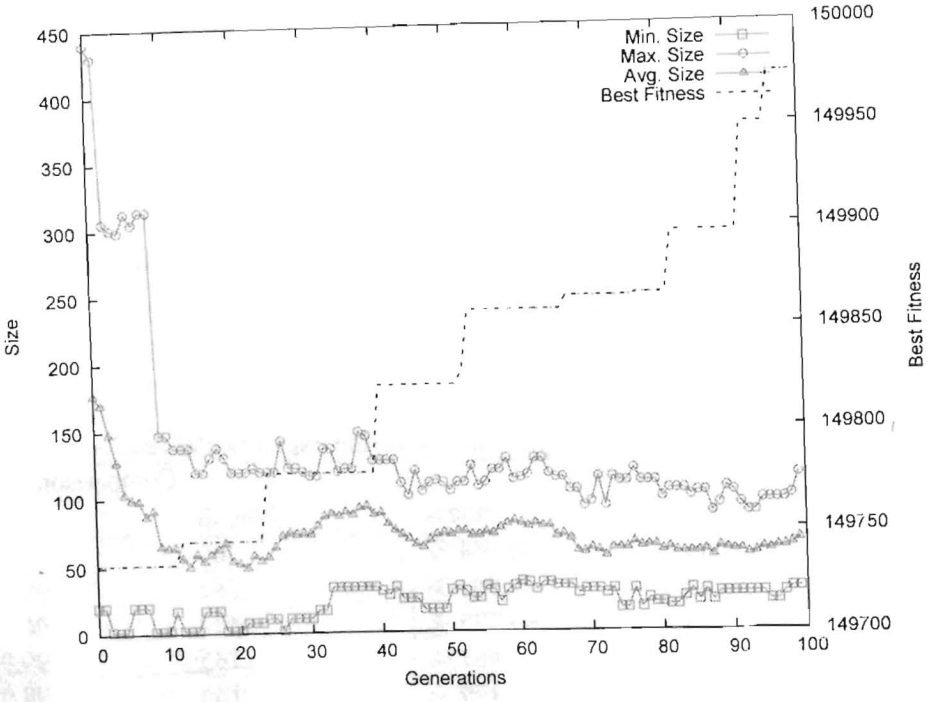


Figure 4.2: Symbolic regression experiment 2: chromosome sizes

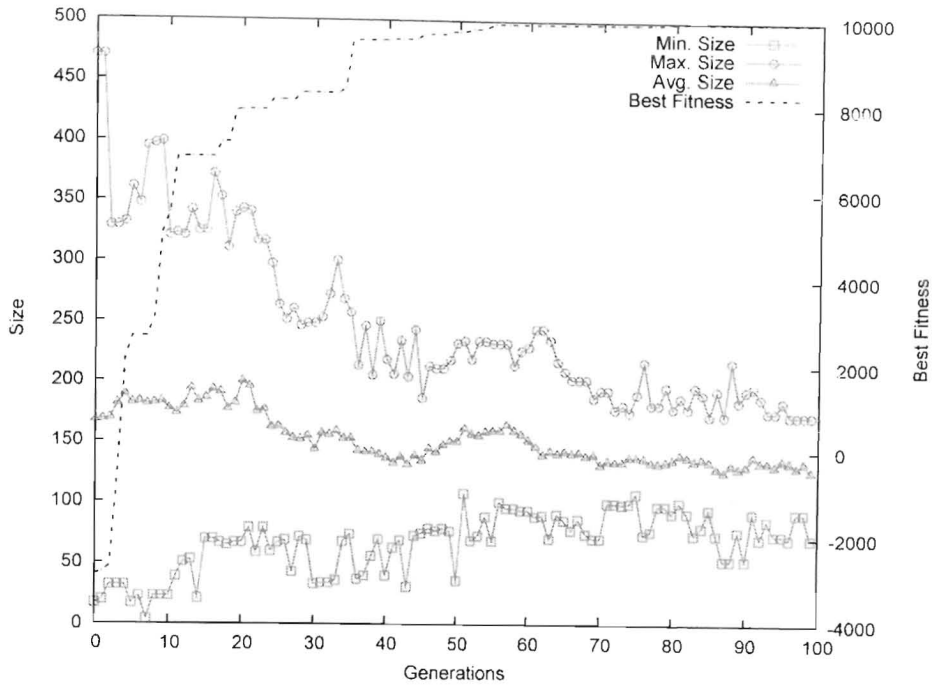


Figure 4.3: Symbolic regression experiment 3: chromosome sizes

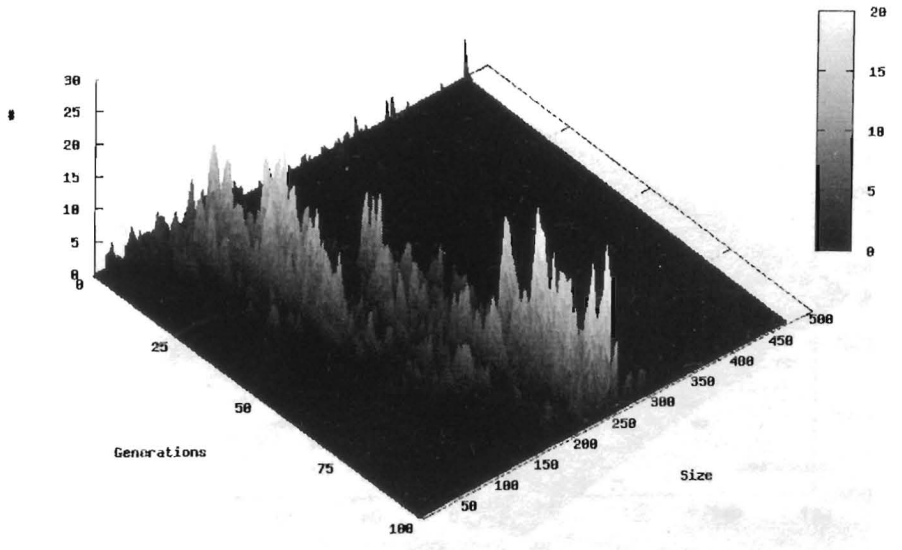


Figure 4.4: Symbolic regression experiment 1: chromosome size in the population

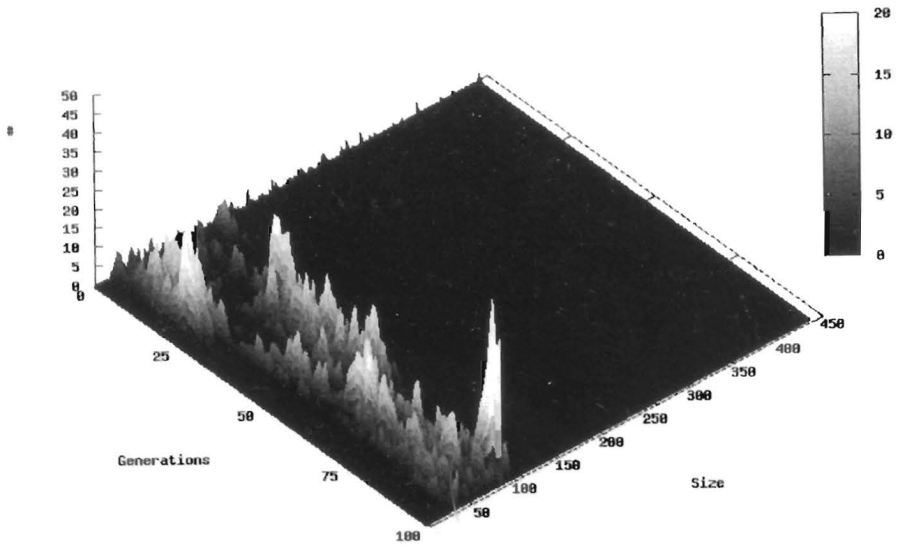


Figure 4.5: Symbolic regression experiment 2: chromosome size in the population

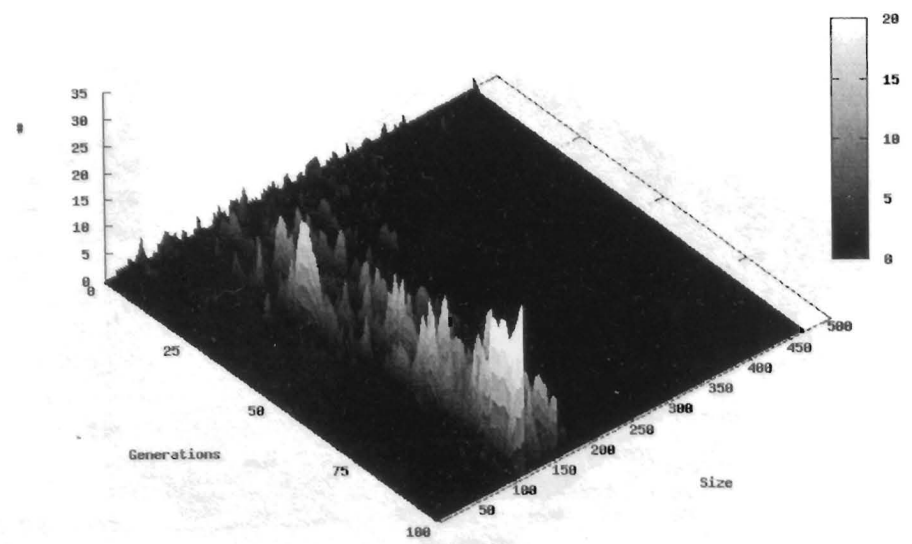


Figure 4.6: Symbolic regression experiment 3: chromosome size in the population

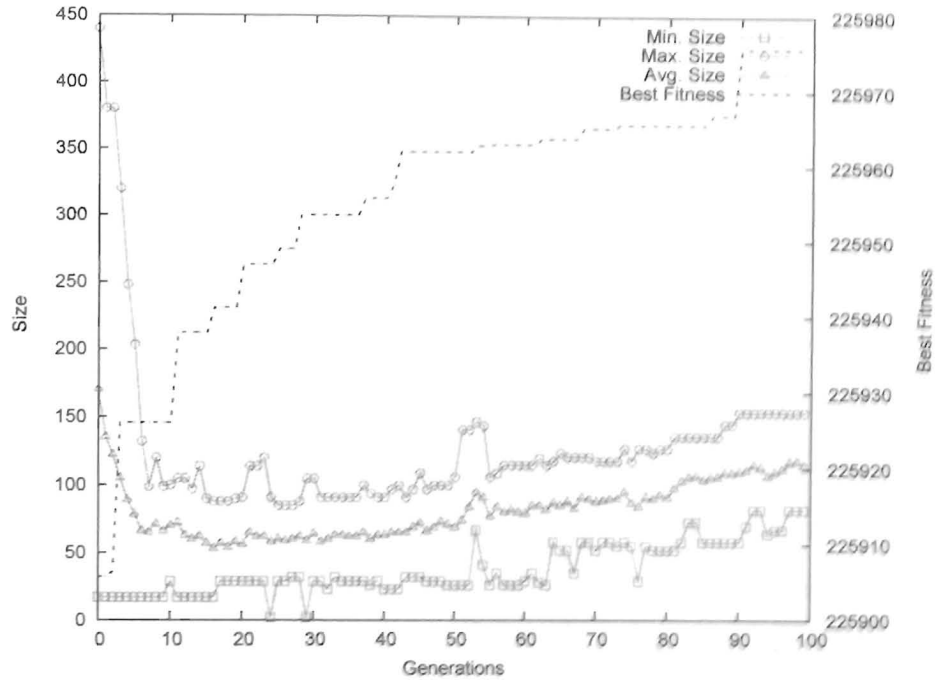


Figure 4.7: Symbolic regression experiment 4: chromosome sizes

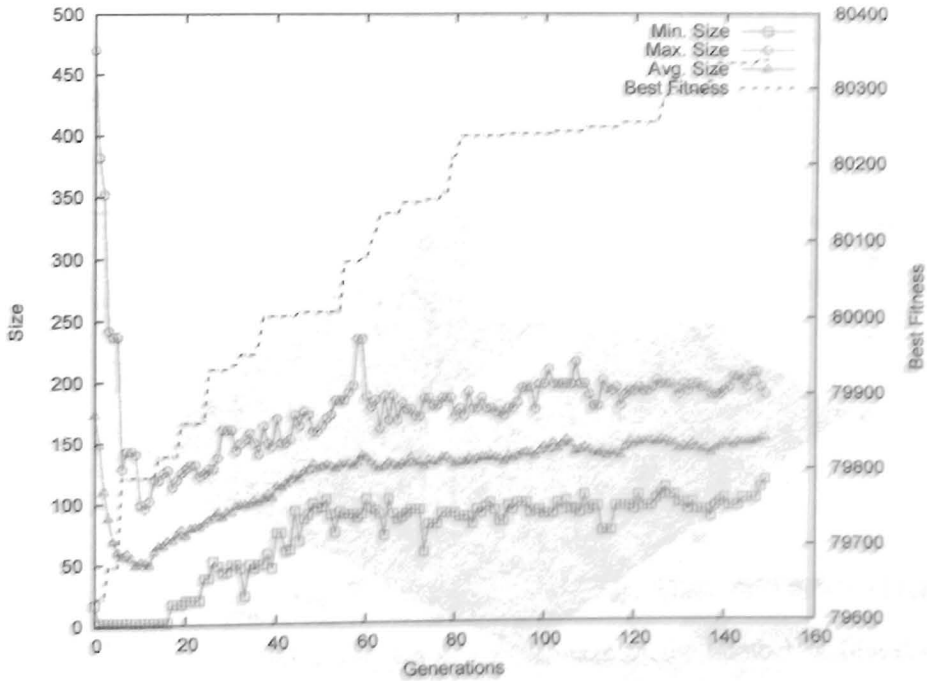


Figure 4.8: Symbolic regression experiment 5: chromosome sizes

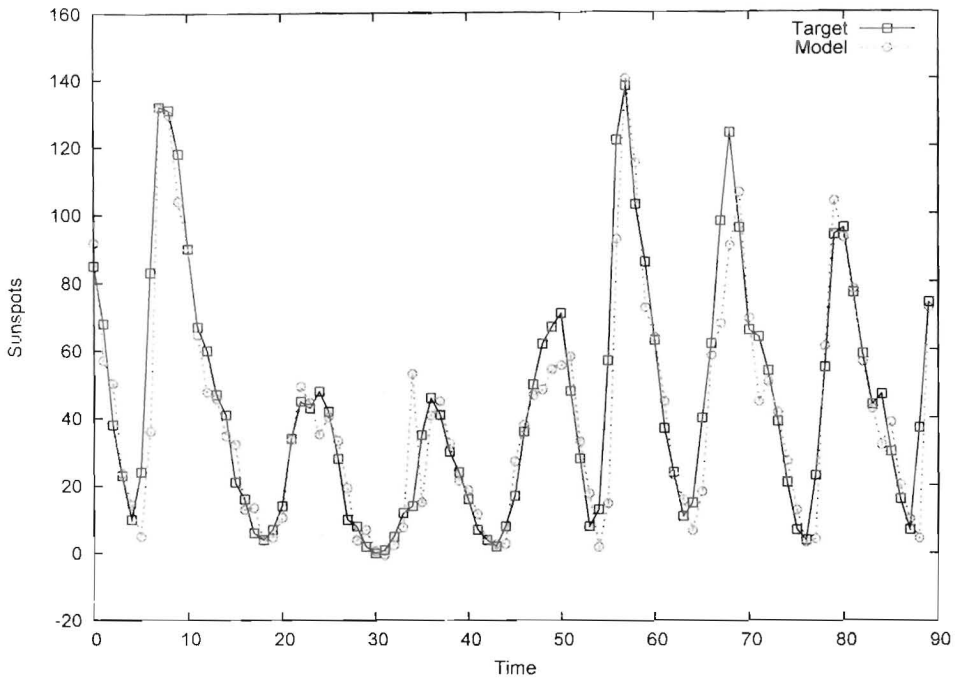


Figure 4.9: Symbolic regression experiment 5: Target vs Model

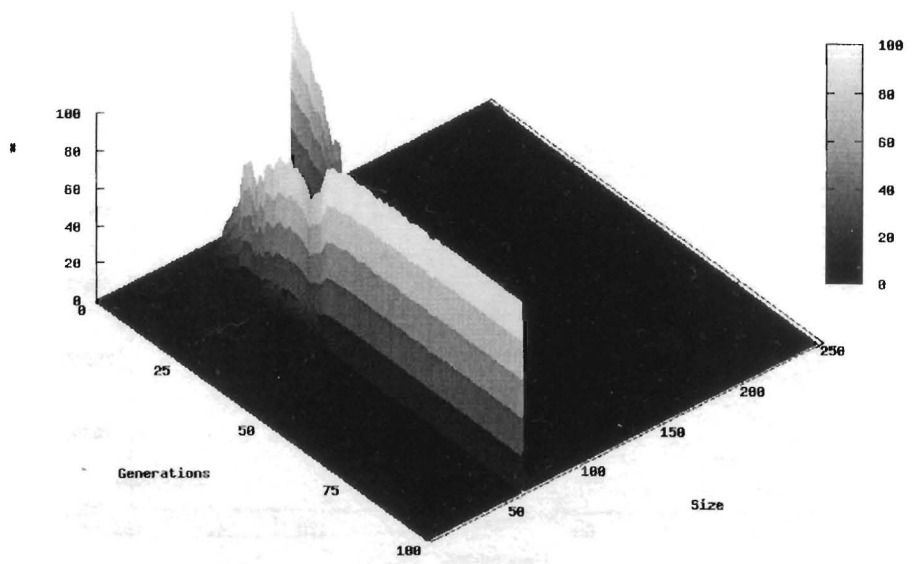


Figure 4.10: Symbolic regression experiment 4: chromosome size in the population

for each generation. The other figures display a surface visualization of the distribution of the chromosome lengths in the population, with respect to the generation number in the run. For the final experiment, the surface plot was omitted because of the rapid convergence to a narrow range of chromosome lengths. Figure 4.9 compares the evolved model's performance to the target data. Since k-fold validation was used, every tenth data point in Figure 4.9 was previously unseen by the model.

The figures show that while the algorithm was optimizing the chromosome length, it initially explored a wide search space, then focused on a band of neighboring chromosome sizes.

A significant result was that the best solutions found using our new operators, evolved better individuals with smaller representations than the PGEP-O system presented in [3] and the canonical GEP algorithm. It is interesting to note that the best chromosomes evolved for the two most difficult problems were significantly smaller than those evolved by the PGEP-O. Specifically, during the second and third experiments, the best evolved individuals were approximately 30% to 33% of the size of the individuals evolved using the PGEP-O methodology. Similarly, in experiments four and five, where our methodology was compared to a distributed canoncial GEP algorithm (based on *Syrah*), our methodology produced results 33% and 30% the size of the alternative's results.

The results of the experiments, as shown in Table 4.1, show that our new operators are significantly more efficient and produced better results for symbolic regression problems. This may have been because our populations were evolving smaller solutions and were able to explore the search space more effectively.

4.2 Classification Results

Table 4.2 shows the results of the classification experiments. These include the chromosome size and the best fitness found, expressed as a percentage of the number of fitness (or test) cases solved. The visualized results and performance of the experiments are shown by Figures 4.11-4.12.

Table 4.2: Summary of classification experimental results

Ours				Canonical GEP			
Total Len.	Genes	Avg Gene Len.	Fitness	Total Len.	Genes	Gene Len.	Fitness
247	8	30.9	81.08 %	770	10	77	80.31 %

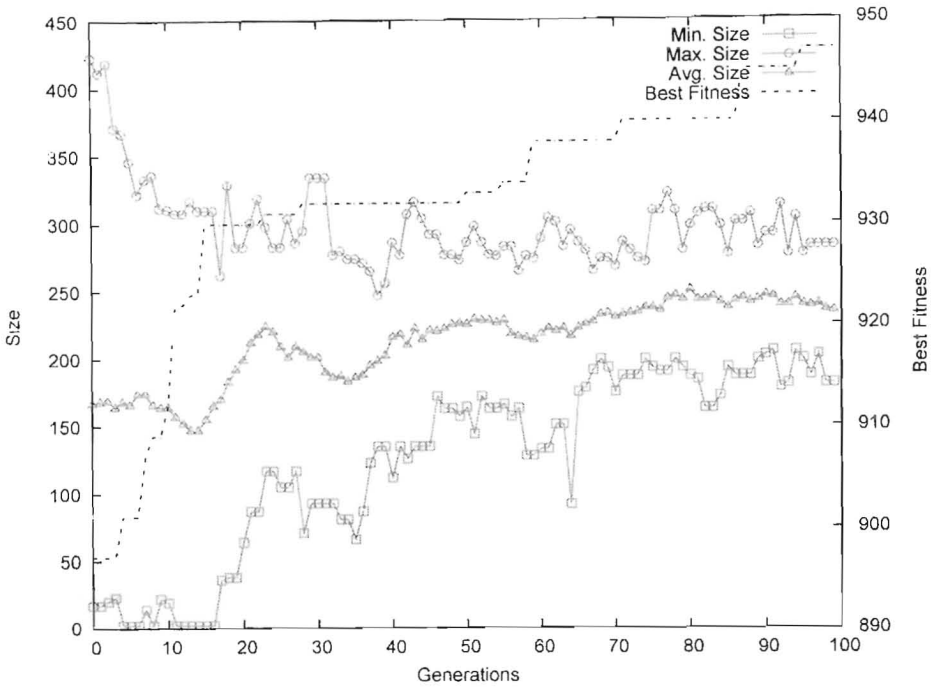


Figure 4.11: LiveDescribe experiment: chromosome sizes

4.2.1 Discussion of Classification Experiments

As stated in chapter 3, the full LiveDescribe data set consisted of approximately 90,000 entries, each with 6 real number variables and grouped into two classes. One of the challenges of this experiment was the computational resources required to evolve candidate solutions.

Both our methodology and canonical GEP evolved individuals with similar performance, with both systems evolving a classifier capable of successfully identifying 80%-81% of the fitness cases. This is a substantial improvement over the original, human written classifier (developed by the CLT at Ryerson [44]), which was able to correctly classify approximately 70% of the fitness cases. Based on discussions with the CLT lab, it is believed that 85% may be the practical limit for identifying non-dialog sections of video using the current variable

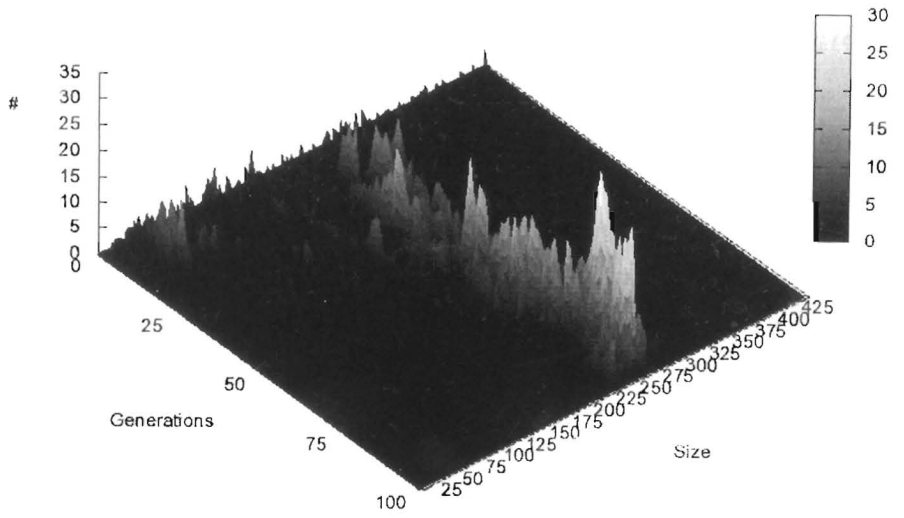


Figure 4.12: LiveDescribe experiment: chromosome size in the population

set. The CLT is currently working to modify their data aquisition software to collect additional parameters.

Examining the solutions evolved by our enhanced algorithm and canonical GEP, it is important to note that our methodology evolved a solution 32.1% the size of the one evolved by the standard algorithm. Since the size of the candidate solution's genome has a direct impact on the evaluation of the fitness cases (and live data, once implemented in the real world), the reduction in representation size may improve the overall performance of the system, even after considering the additional computation requirements of our new operators.

The small number of classes in this experiment may have been a possible limitation. With only two possible classes, the evolutionary process may not have been significantly challenged. However, it is felt that the number of test cases may have offset this. In the future, more complex classification problems should be investigated.

What the summary of results do not show is the number of additional runs (and thus processing time) required to evaluate different values for the head domain length and number of

genes for the canonical GEP algorithm that was used for comparison.

4.3 Parameter Optimization Results

Table 4.3 shows a summary of the results of parameter optimization experiments. The summary shows the maximum function value found, the average gene length (static for canonical GEP) and the total genome size. The visualized results and performance of the experiments are shown by Figures 4.13-4.22.

Table 4.3: Summary of parameter optimization experimental results

Exper.	Ours			Canonical GEP		
Number	Total Len.	Avg. Gene Len.	Maximum	Total Len.	Gene Len.	Maximum
1	105	35	78.30	231	77	78.51
2	10	5	3904.62	184	92	3902.40
3	25	5	25	385	77	25
4	426	14.2	1233.87	780	26	1125.61
5	22	11	499.002	94	47	499.002

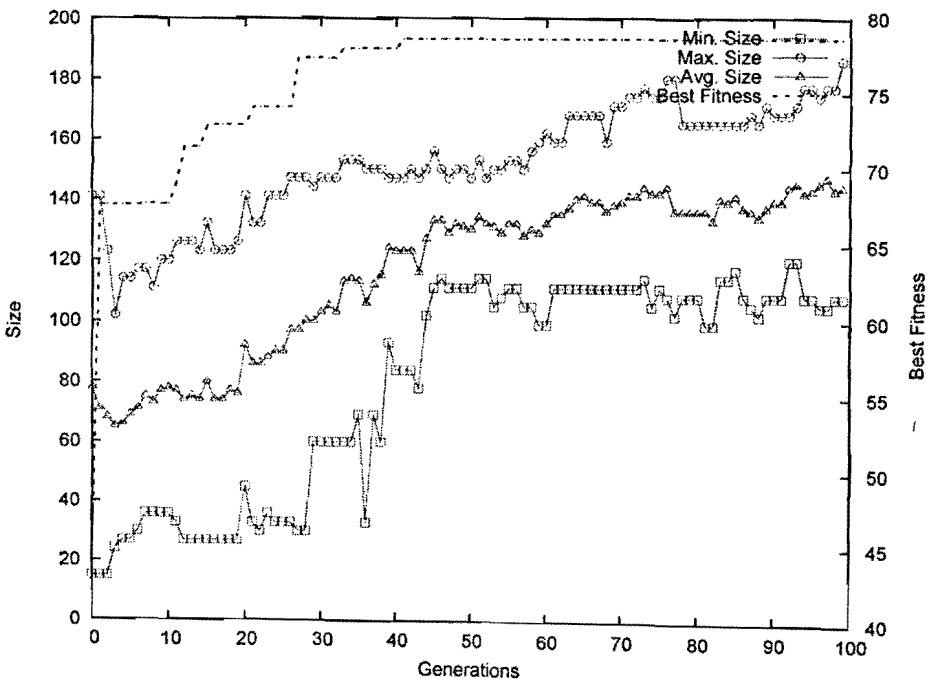


Figure 4.13: Parameter optimization experiment 1: chromosome sizes

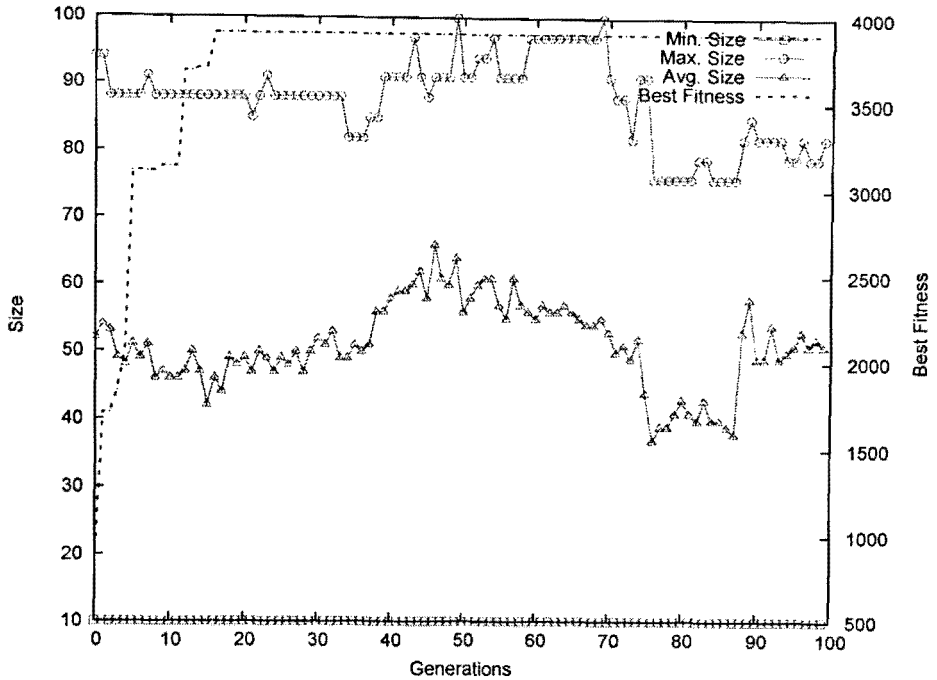


Figure 4.14: Parameter optimization experiment 2: chromosome sizes

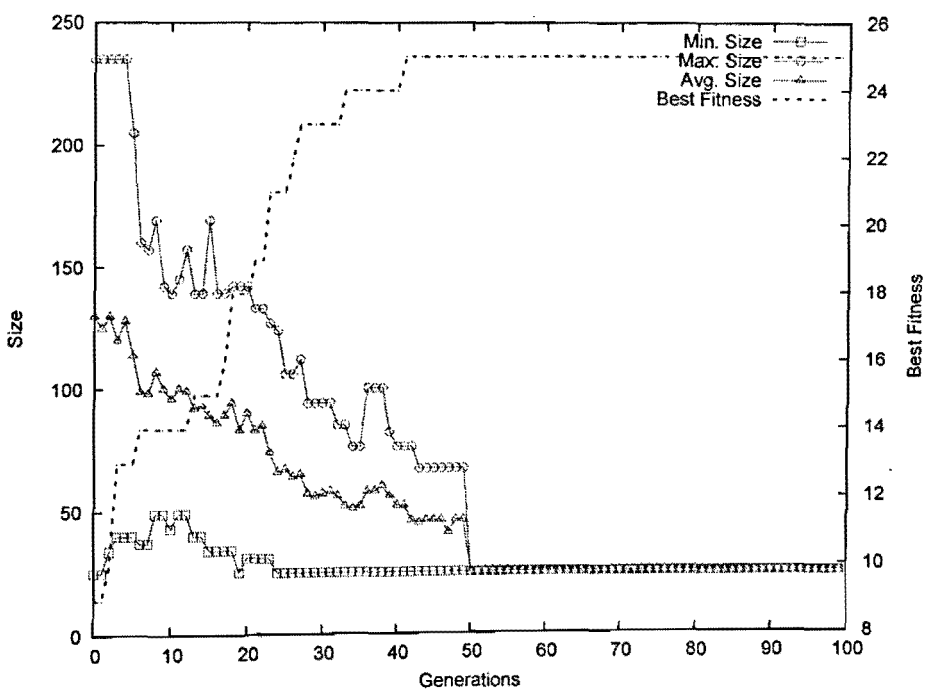


Figure 4.15: Parameter optimization experiment 3: chromosome sizes

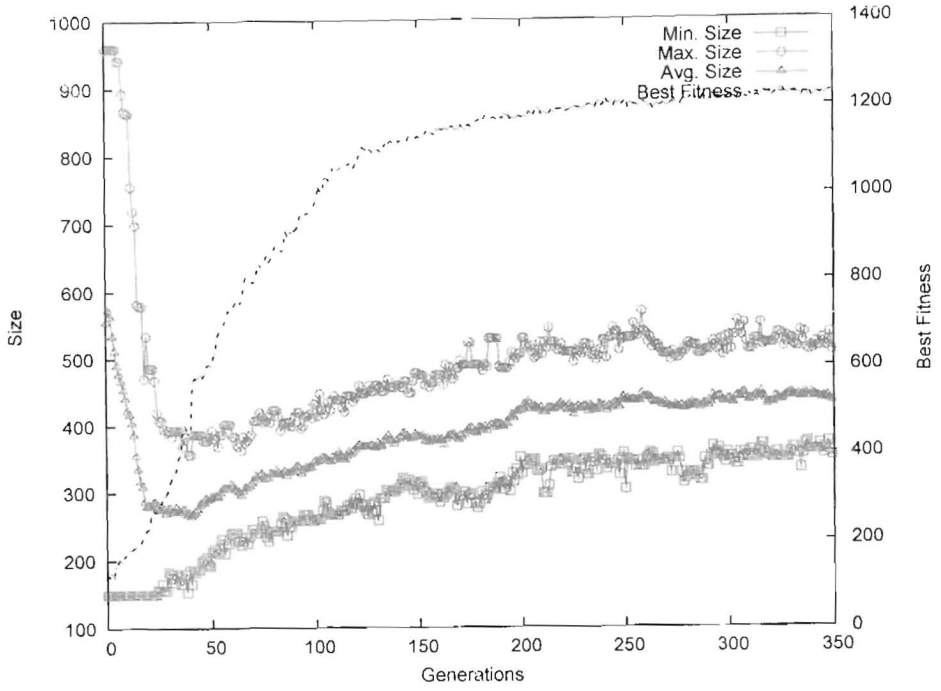


Figure 4.16: Parameter optimization experiment 4: chromosome sizes

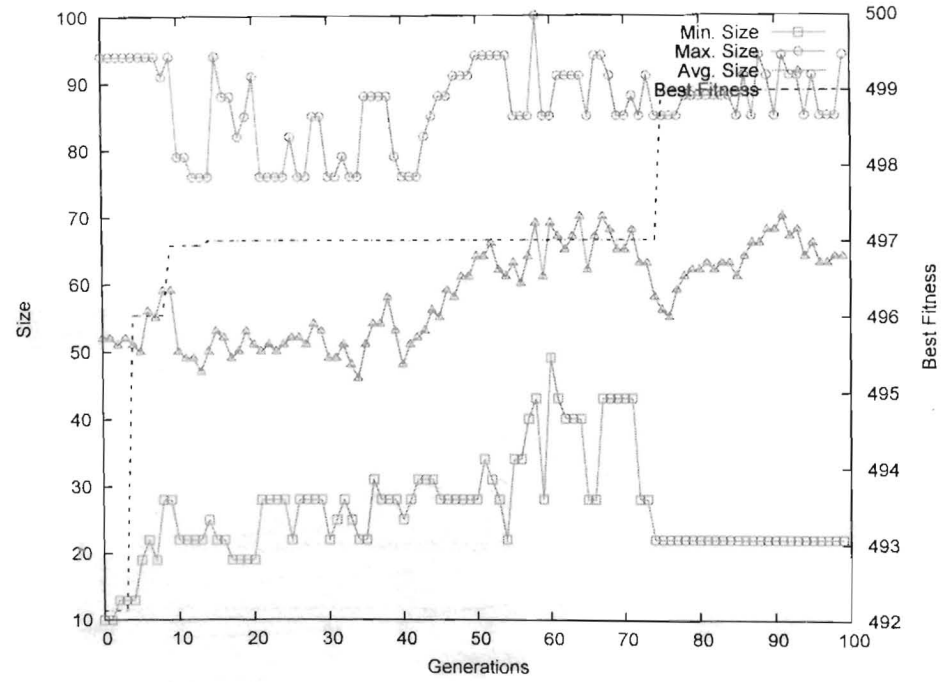


Figure 4.17: Parameter optimization experiment 5: chromosome sizes

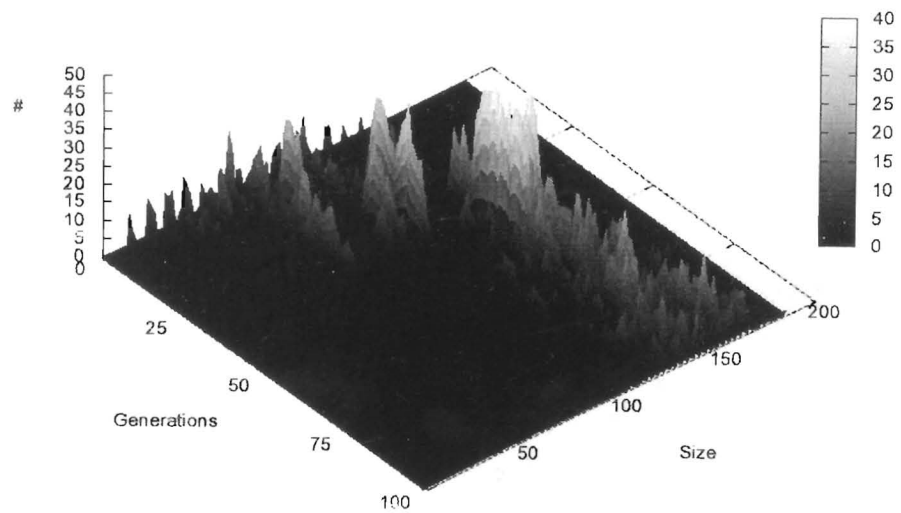


Figure 4.18: Parameter optimization experiment 1: chromosome size in the population

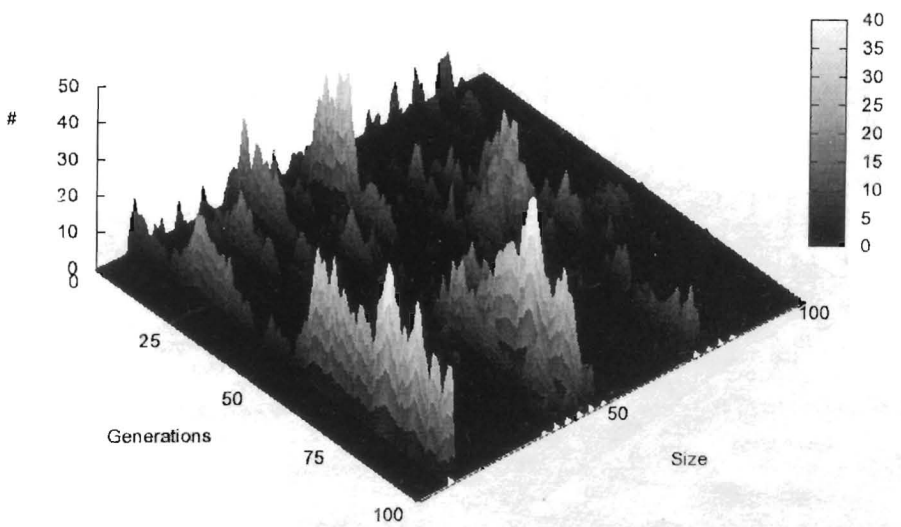


Figure 4.19: Parameter optimization experiment 2: chromosome size in the population

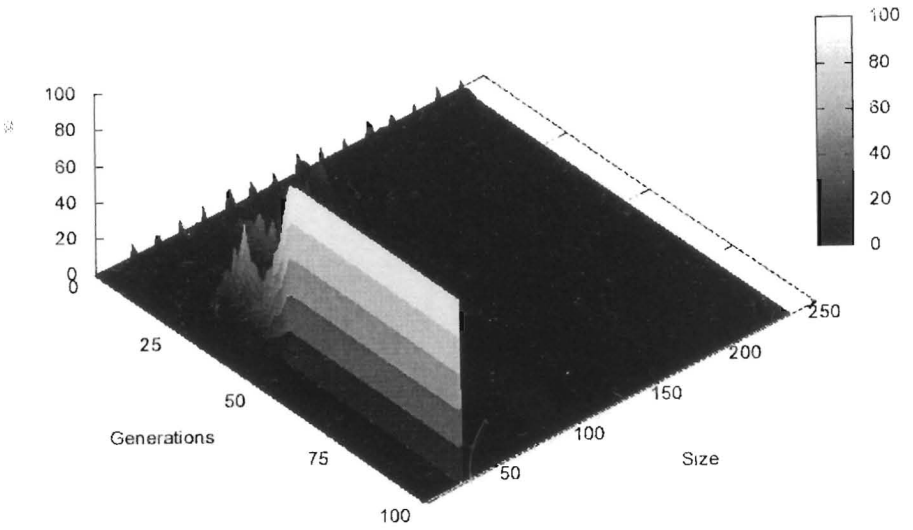


Figure 4.20: Parameter optimization experiment 3: chromosome size in the population

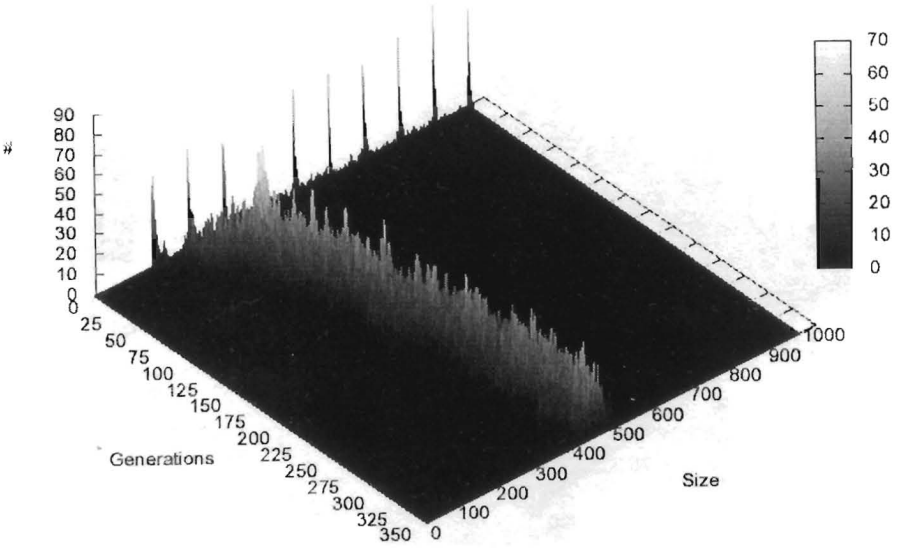


Figure 4.21: Parameter optimization experiment 4: chromosome size in the population

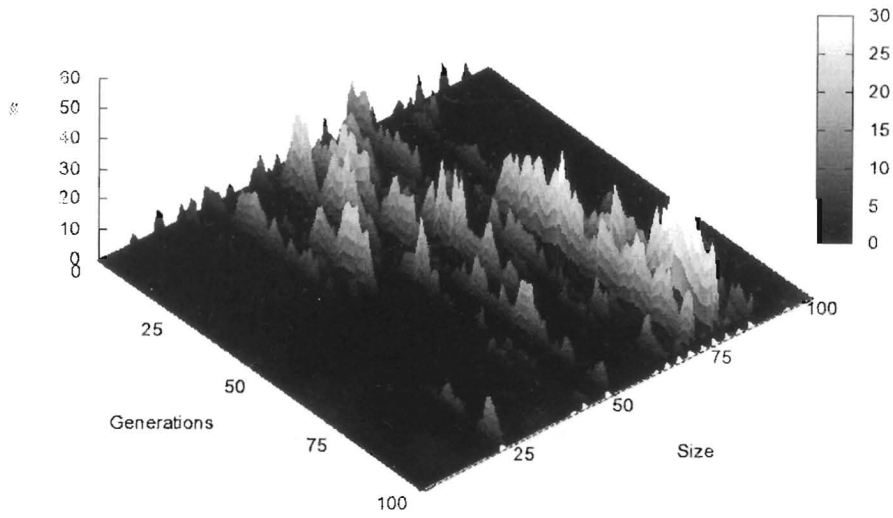


Figure 4.22: Parameter optimization experiment 5: chromosome size in the population

4.3.1 Discussion of Parameter Optimization Experiments

The results of the parameter optimization experiments show that both our methodology and canonical GEP are effective at evolving either optimal or near-optimal solutions to the problems in the De Jong test suite. As seen in the previous series of experiments, our enhancements enabled the algorithm to consistently evolve solutions which were significantly smaller than those evolved by canonical GEP.

The solutions evolved by our enhanced algorithm in experiments two and three were remarkably smaller than those found by canonical GEP. Specifically, they were 5.4% and 6.5% the size of those found by standard GEP.

Both methodologies had difficultly with the high-dimension problem found in parameter optimization experiment 4. However, our enhanced GEP algorithm evolved a slightly better result and had a representation size 54.6% the size of the one evolved by the standard algorithm. It is believed that the difficultly of this problem and the inability of the algorithm to locate the optimal parameter values contributed to the evolved size of the genome. Similarly, the

numerical results of experiment 1 were comparable, but the solutions evolved using the HIS operator and our other enhancements were 45.5% the size of standard GEP's solutions.

The chromosome sizes evolved during final parameter optimization experiment were closer to what we had observed during the Symbolic Regression and Classification experiments, with our evolved solutions being approximately 23.4% the size of those evolved by canonical GEP. In this case, both methodologies successfully found the maximum value of Shekel's fox-holes.

All of the parameter optimization experiments have shown that our enhancements retained GEP's problem solving ability while allowing it to evolve smaller genomes. While the De Jong functions have been reported [48] to not be an effective test set, they have been repeatedly shown to provide a good metric of the effectiveness of algorithms for a broad range of optimization problems.

A possible limitation is that it is not currently possible to use the ACS mutation operator with our existing experimental setup. Since we have not used Automatically Defined Functions (ADFs) [1], we must use a fixed number of genes - one per parameter requiring optimization. While we were still able to obtain good results, we can only speculate that using ADFs and allowing the number of "normal" genes to evolve (as they do in the symbolic regression and classification experiments), would enhance the solutions of more difficult parameter optimization problems.

4.4 General Discussion

Reviewing the results of our experiments, we see that our enhancements to the GEP algorithm consistently produced smaller solutions (sometimes significantly so) than canonical GEP. Since the representation size of a genome has a direct impact on the evaluation of the fitness cases, the reduction in representation size may improve the overall performance of the system, even after considering the additional computation requirements of our new operators. This was indirectly observed during the classification experiments while waiting for the two methodologies to complete their evolutionary runs. When our enhanced algorithm was running, it was notice-

ably faster than when the standard GEP algorithm was processing the same problem.

The tuning of the number of genes and the head size of each gene was an implicit part of our GEP run and, thus, we did not require separate clients for optimization. This reduced the overall computational resources required to evolve solutions.

For all of the parameter optimization experiments the ACS mutation operator was disabled and thus, we were unable to evaluate its potential effectiveness for this class of problems. The operator was disabled because of the evaluation method used. Since our GEP implementation did not use ADFs, it required one gene per parameter to optimize. It is possible that if we implemented automatically defined functions and used the ACS mutation operator to evolve the number of "normal" genes, we would see different results.

The decision to randomly initialize the genes that were inserted during the ACS mutation phase appears successful. However, it would be interesting to investigate the use of gene cloning, or other methods, in the future.

We observed that the insertion point in the ACS mutation operator for classification and symbolic regression problems was not important because we used a commutative linking function during testing. The insertion point, however, may have been significant because of the way the gene would mix within the population during recombination. Additionally, since the Gene Transposition operator was used, good genes could be reordered within the chromosome. Had we used a non-commutative linking function or homeotic (ADF) genes, the insertion location could have had a greater impact.

Based on the results of our experiments, our new operators were able to successfully self-adaptively tune the two critical parameters of the GEP algorithm, the head domain length and the number of genes. While our new operators have additional computational costs associated with them, it is believed that the additional operator execution times are offset by the shorter time required to evaluate the fitness functions, because of the smaller representations it evolved.

Our new recombination operators have also been empirically shown to permit crossbreeding and speciation within a single GEP population. Additionally, our operators have been shown to be effective in a distributed environment. However, additional research into the effects of our

operators on migration is required.

Chapter 5

Conclusion and Future Work

This thesis presented novel enhancements to the Gene Expression Programming algorithm that enabled flexible genome representations, endowed self-adaptive characteristics, assisted with maintaining diversity within a population and enhanced the parallelization of the algorithm. In particular, the enhancements addressed issues of evolvability, crossbreeding and speciation, parameter tuning and parallelization in canonical GEP.

Through a series of experiments that used an assortment of problem classes, including symbolic regression, classification and parameter optimization, we have shown that our proposed methodology produced better results and, generally, smaller genome representations than the canonical GEP algorithm and the PGEP-O system [3] (for symbolic regression).

Specifically, the contributions presented in this work were:

1. Creation of a new transposition operator, the *Head Insertion Sequence* (HIS), which self-adaptively tunes the head domain length of a gene.

2. Development of a new mutation operator, the *Adaptive Chromosome Size* (ACS) mutation, which mutates the number of genes in an individual to tune the gene count parameter.

3. Addition of new GEP recombination operators to permit structurally dissimilar individuals to interact. This removed the structural constraints imposed when transferring an individual from one population to another and permitted both crossbreeding and speciation.

Our enhancements to the GEP algorithm also simplified its use, by implicitly tuning the head domain length and number of genes throughout an evolutionary run. By removing the

need to set these two critical GEP parameters prior to executing a run, the level of "expert knowledge" required to use GEP is reduced and allows EC novices to use the algorithm more effectively.

The simplification of the algorithm's configuration and the implicit parameter tuning of the two critical parameters are still subject to the concept of "No Free Lunch" [35]. The "No Free Lunch" theorem [35] states that without *a priori* knowledge of a problem, all potential solution methods are equal. While the values of the parameters evolved during a run may not be optimal for all problem types, they are frequently "good enough" and "No Free Lunch" is partially offset by the ease of using the new algorithm. This was seen during our experimental verification of the algorithm and when comparing our methodology to canonical GEP. To determine the GEP experimental baselines, several runs with different head domain length and number of gene parameter values were required, to obtain usable results. Comparatively, with our enhanced algorithm we only needed to start a run sequence and let the algorithm evolve the parameters.

While our enhancements to the GEP algorithm have proven to be successful, they are not without costs and limitations. Since we have added extra operators to enable our meta-evolution of the parameters, we also have added additional computational overhead. In particular, the ACS mutation operator has significant overhead when it generates a new gene from random elements. The overhead associated with the new operators may be partially offset by the reduced size of the solution representations (as experienced during our trials), but further experimentation and analysis are required to confirm this.

Another side effect of our self-adaptive method is that we have increased the search space available to the algorithm. This is both a benefit and a liability, since the algorithm can traverse the entire space defined by any combination of head domain length and number of genes. This allows the algorithm to find novel solutions, but also increases the number of potential solutions dramatically, possibly increasing the search time and allowing the algorithm to get stuck in at a non-optimal solution.

When developing the enhancements to the GEP algorithm, the possibly of introducing bloat, or the excessive creation of introns to protect a genome's functionality, was a major concern.

By eliminating the fixed chromosome size (which was necessary to remedy the issues we saw with GEP), the potential for the genome representation and size to grow unchecked became a possibility, even with the parameter tuning inherent in the new operators. One possible reason for not observing bloat is because the HIS Transposition operator, which is responsible for tuning the head size, restructures the genome by adding sections from one domain to another instead of simply inserting or deleting material. This does not account for the effect of the ACS mutation operator, which mutates the number of genes in a chromosome. However, the selection pressure from the Tournament Selection with Elitism selection method may have provided resistance to unnecessary gene additions. It is possible that in more difficult problems (that require longer runs or larger datasets), we may begin to observe bloat and need to take steps to measure and constrain it.

Related to the previous topic of bloat and introns, is the matter of genetic diversity within a population. Our current research did not include any specific mechanisms to measure the diversity of individuals within a population (either in a single population or distributed multi-population setting), but the genome length statistics, recorded during the experiments, can be used as a simple metric. Using the surface plots of the chromosome lengths (found in chapter 4) we can suppose that our methodology maintains a level of genetic diversity throughout a run. While the populations were initially very diverse and chaotic, as the runs progressed the outliers were reduced and a narrower band of chromosome sizes (and thus diversity) was maintained.

Overall, our enhancements have been shown to be effective at addressing the issues of evolvability, crossbreeding and speciation, parameter tuning and parallelization in the canonical GEP algorithm.

5.1 Future Work

Though our enhancements have been effective, there is still work that can be done to further our understanding of them, their relationship and application to Evolutionary Computation in general, and the workings of the GEP algorithm itself.

A detailed study of the effects of our enhancements on the levels of genetic diversity in a population would aid in understanding the mechanisms that make the operators effective. Additionally, applying the "Nonsynonymous to Synonymous Substitution Ratio (Ka/Ks)" [49] to study the rate of evolution, in conjunction with a diversity study, could show where further improvements could be made in the GEP algorithm.

Applying our enhancements to Automatically Defined Functions (ADFs) in GEP could potentially provide interesting results and bears further investigation. This could be particularly useful for difficult or complex parameter optimization problems, since, when using GEP-PO, the number of genes must always equal the number of parameters being optimized. Using ADFs would allow the number of normal genes to be adaptively tuned using the ACS mutation operator.

Further research into the potential of unrestrained chromosome growth, or bloat, and selection pressure in our enhanced GEP algorithm would be interesting, as we did not observe significant bloat during our experiments. In evolutionary computation, any algorithm or representation that allows unrestrained growth and yet demonstrates resistance to bloat warrants further investigation.

The impact of our operators on migration and the exchange of genetic material in a distributed setting requires further study. In particular, a thorough examination our system when running in a distributed, multi-island settings with different connection topologies and migration strategies would be useful for determining the optimal configuration (if possible).

While the enhancements presented in this work enabled crossbreeding and the evolution of different species within a population, we did not specifically implement any niching methods. This could prove to be an interesting avenue of exploration in the future, as it could enhance the algorithm's performance with multi-modal problems.

Finally, adapting our enhancements to neuroevolution, or the evolution of neural networks, using GEP (such as the GEP-nets algorithm [7]) has great potential. This is because our enhancements could permit size and structure changes to the evolved neural networks, allowing a more dynamic and complicated structure to be evolved.

Bibliography

[1] C. Ferreira, "Gene expression programming: A new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.

[2] K. A. De Jong, "Analysis of the behavior of a class of genetic adaptive systems," Ph.D. dissertation, University of Michigan, 1975. [Online]. Available: http://hdl.handle.net/2027.42/4507

[3] H.-H. Park, A. Grings, M. dos Santos, and A. Soares, "Parallel hybrid evolutionary computation: Automatic tuning of parameters for parallel gene expression programming," *Appl Math Comput (New York)*, 2008.

[4] K. Zhang, S. Sun, and H. Si, "Prediction of retention times for a large set of pesticides based on improved gene expression programming," in *GECCO '08: Proceedings of the 10th annual conference on Genetic and evolutionary computation.* New York, NY, USA: ACM, 2008, pp. 1725–1726.

[5] Z. Xie, X. Li, B. Di Eugenio, P. C. Nelson, W. Xiao, and T. M. Tirpak, "Using gene expression programming to construct sentence ranking functions for text summarization," in *COLING '04: Proceedings of the 20th international conference on Computational Linguistics.* Morristown, NJ, USA: Association for Computational Linguistics, 2004, p. 1381.

[6] J. Venter and A. Hardy, "Generating plants with gene expression programming," in *AFRIGRAPH '07: Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa.* New York, NY, USA: ACM, 2007, pp. 159–167.

[7] C. Ferreira, *Gene Expression Programming - Mathematical Modeling by an Artificial Intelligence*, 2nd ed. Springer-Verlag, 2006.

[8] M. Ostaszewski, P. Bouvry, and F. Seredynski, "Multiobjective classification with mogep: an application in the network traffic domain," in *GECCO '09: Proceedings of the 11th Annual conference on Genetic and evolutionary computation.* New York, NY, USA: ACM, 2009, pp. 635–642.

[9] J. Yin, L. Huo, L. Guo, and J. Hu, "Short-term load forecasting based on improved gene expression programming," in *Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on*, June 2008, pp. 5647–5650.

[10] J. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* The MIT Press, 1992.

[11] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley Professional, 1989.

[12] J. Reisinger, K. O. Stanley, and R. Miikkulainen, "Towards an empirical measure of evolvability," in *GECCO '05: Proceedings of the 2005 workshops on Genetic and evolutionary computation.* New York, NY, USA: ACM, 2005, pp. 257–264.

[13] K. O. Stanley, "Efficient evolution of neural networks through complexification," Ph.D. dissertation, The University of Texas at Austin, August 2004.

[14] W. R. W. Heitor S. Lopes, "Egipsys: An enhanced gene expression programming approach for symbolic regression problems," *Int. J. Appl. Math. Comput. Sci*, vol. 14, no. 3, pp. 375–384, 2004.

[15] J. Yue, T. Chang-jie, Z. Hai-chun, L. Chuan, C. Yu, W. Jiang, and W. Dong-lei, "Adaptive gene expression programming algorithm based on cloud model," in *BioMedical Engineering and Informatics, 2008. BMEI 2008. International Conference on*, vol. 1, May 2008, pp. 226–230.

[16] C. Tang, L. Duan, J. Peng, H. Zhang, and Y. Zong, "The strategies to improve performance of function mining by gene expression programming: Genetic modifying, overlapped

gene, backtracking and adaptive mutation." in *Proceedings of the 17th Data Engineering Workshop*, 2006.

[17] O. M. Shir and T. Back, *Algorithmic Bioprocesses*. Springer Berlin Heidelberg, 2009, ch. Niching Methods: Speciation Theory Applied for Multi-modal Function Optimization, pp. 705–729. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-88869-7_35

[18] J. H. Holland, *Adaptation in natural and artificial systems. an introductory analysis with applications to biology, control and artificial intelligence*. University of Michigan Press, 1975.

[19] E. Bautu, A. Bautu, and H. Luchian, "Adagep - an adaptive gene expression programming algorithm," *Symbolic and Numeric Algorithms for Scientific Computing, 2007. SYNASC. International Symposium on*, pp. 403–406, Sept. 2007.

[20] M. Gorges-Schleuter, "Explicit parallelism of genetic algorithms through population structures," *Parallel Problem Solving from Nature*, pp. 150–159, 1991. [Online]. Available: http://dx.doi.org/10.1007/BFb0029746

[21] I. Hidalgo, F. De Vega, J. Lanchares, and D. Lombrana, "Is the island model fault tolerant?" in *9th Annual Genetic and Evolutionary Computation Conference, GECCO 2007*, London, 2007.

[22] E. Cantu-Paz and D. Goldberg, "Efficient parallel genetic algorithms: Theory and practice," *Comput. Methods Appl. Mech. Eng.*, vol. 186, no. 2-4, pp. 221–238, 2000.

[23] J. Berntsson and M. Tang, "Dynamic optimization of migration topology in internet-based distributed genetic algorithms," in *GECCO 2005 - Genetic and Evolutionary Computation Conference*, B. H.G., O. U.M., A. D., B. W., B. C., B. E.W., C.-P. E., D. D., D. K., and et al, Eds., Washington, D.C., 2005, pp. 1579–1580.

[24] E. Cantu-Paz, *Efficient and Accurate Parallel Genetic Algorithms*. Springer, 2000.

[25] E. Alba and J. Troya, "Analyzing synchronous and asynchronous parallel distributed genetic algorithms," *Future Gener Comput Syst*, vol. 17, no. 4, pp. 451–465, 2001.

[26] Z. Skolicki and K. De Jong, "The influence of migration sizes and intervals on island models," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2005.

[27] Z. Skolicki and K. De Jong, "The importance of a two level perspective for island model design," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2008.

[28] S.-K. Oh, C. T. Kim, and J.-J. Lee, "Balancing the selection pressures and migration schemes in parallel genetic algorithms for planning multiple paths," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, vol. 4, 2001, pp. 3314–3319 vol.4.

[29] S.-C. Lin, I. Punch, W.F., and E. Goodman, "Coarse-grain parallel genetic algorithms: categorization and new approach," in *Parallel and Distributed Processing, 1994. Proceedings. Sixth IEEE Symposium on*, Oct 1994, pp. 28–37.

[30] E. Alba and J. Troya, "Influence of the migration policy in parallel distributed gas with structured and panmictic populations," *Appl Intell*, vol. 12, no. 3, pp. 163–181, 2000.

[31] Y. Lin, H. Peng, and J. Wei, "A niching gene expression programming algorithm based on parallel model," in *Lecture Notes in Computer Science*, vol. 4847 LNCS, Guangzhou, 2007, pp. 261–270.

[32] J. Wu, C. Tang, T. Li, S. Qiao, Y. Jiang, and S. Ye, "Parallel multi-objective gene expression programming based on area penalty," in *International Conference on Computer Science and Information Technology*, 29 2008-Sept. 2 2008, pp. 264–268.

[33] Q. Liu, T. Li, C. Tang, Q. Liu, C. Li, and S. Qiao, "Multi-population parallel genetic algorithm for economic statistical information mining based on gene expression programming," in *Natural Computation, 2007. ICNC 2007. Third International Conference on*, vol. 3, Aug. 2007, pp. 461–465.

[34] X. Du, L. Ding, and L. Jia, "Asynchronous distributed parallel gene expression programming based on estimation of distribution algorithm," in *Natural Computation, 2008. ICNC '08. Fourth International Conference on*, vol. 1, Oct. 2008, pp. 433–437.

[35] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Trans Evol Comput*, vol. 1, no. 1, pp. 67–82, 1997.

[36] Y. Ho and D. Pepyne, "Simple explanation of the no-free-lunch theorem and its implications," *J. Optim. Theory Appl.*, vol. 115, no. 3, pp. 549–570, 2002.

[37] A. Eiben, Z. Michalewicz, M. Schoenauer, and J. Smith, "Parameter control in evolutionary algorithms," *Stud. Comput. Intell.*, vol. 54, pp. 19–46, 2007.

[38] T. Bck, "Self-adaptation in genetic algorithms," in *Proceedings of the First European Conference on Artificial Life*. MIT Press, 1992, pp. 263–271.

[39] M. H.-j. LI De-yi and S. Xue-mei, "Membership clouds and membership cloud generators," *Journal of Computer Research and Development*, pp. 15–20., 1995.

[40] C. Ferreira, "Questions and answers from personal correspondence," September 2009. [Online]. Available: http://www.gene-expression-programming.com/Q&A03.asp

[41] J. Branke, A. Kamper, and H. Schmeck, "Distribution of evolutionary algorithms in heterogeneous networks," *Lect. Notes Comput. Sci.*, vol. 3102, pp. 923–934, 2004.

[42] R. McKay, H. X. Nguyen, J. R. Cheney, M. Kim, N. Mori, and T. H. Hoang, "Estimating the distribution and propagation of genetic programming building blocks through tree compression," in *Proceedings of GECCO 2009*, Sigevo. ACM, 2009.

[43] T. W. Anderson, *The statistical analysis of time series*. Wiley New York,, 1971.

[44] Livedescribe video description software. The Center for Learning Technology at Ryerson University. [Online]. Available: http://www.livedescribe.com/

[45] H. Rosenbrock, "An automatic method for finding the greatest or least value of a function." *Comp. J.*, vol. 3, pp. 175–184., 1960.

[46] J. Shekel, "Test functions for multimodal search techniques," in *Fifth Annual Princeton Conference on Information Science and Systems.*, 1971.

[47] J. Alami, A. E. Imrani, and A. Bouroumi, "A multipopulation cultural algorithm using fuzzy clustering," *Applied Soft Computing*, vol. 7, no. 2, pp. 506 – 519, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/B6W86-4MJC1VF-2/2/3c9b90ea39aab93a935d81511d0614c2

[48] D. Whitley, K. Mathias, S. Rana, and J. Dzubera, "Building better test functions," *Proc. Sixth International Conference on Genetic Algorithms*, vol. -, pp. 239–246, 1995.

[49] T. Hu and W. Banzhaf, "Nonsynonymous to synonymous substitution ratio ka/ks: Measurement for rate of evolution in evolutionary computation," *Lecture Notes in Computer Science*, pp. 448–457, 2008.