PROBABILISTIC ADAPTIVE MAPPING DEVELOPMENTAL GENETIC PROGRAMMING

by

Garnett Carl Wilson

Submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

at

Dalhousie University
Halifax, Nova Scotia
March 2007

*** Signature page is supplied by FGS at time of PhD defense. ***

DALHOUSIE UNIVERSITY

DATE: March 7, 2007

AUTHOR: Garnett Carl Wilson

TITLE: PROBABILISTIC ADAPTIVE MAPPING DEVELOPMENTAL GENETIC PROGRAMMING

DEPARTMENT OR SCHOOL: Faculty of Computer Science

DEGREE: Ph.D. CONVOCATION: May YEAR: 2007

To Michelle with love,

for making me happier
than I ever thought I could be.

Table of Contents

List of Figures

List of Tables

Abstract

Developmental Genetic Programming (DGP) algorithms explicitly enable the search space for a problem to be divided into genotypes and corresponding phenotypes. The two search spaces are often connected with a genotype-phenotype mapping (GPM) intended to model the biological genetic code, where current implementations of this concept involve evolution of the mappings along with evolution of the genotype solutions. This work presents the Probabilistic Adaptive Mapping DGP (PAM DGP) algorithm, a new developmental implementation that provides research contributions in the areas of GPMs and coevolution. The algorithm component of PAM DGP is demonstrated to overcome coevolutionary performance problems as identified and empirically benchmarked against the latest competing Adaptive Mapping algorithm with both algorithms using the same (non-redundant) mapping encoding process. Having established that PAM DGP provides a superior algorithmic framework given equivalent mapping and genotype structures for the individuals, a new adaptive redundant mapping is incorporated into PAM DGP for further performance enhancement and closer adherence to developmental modeling of the biological code. PAM DGP with two mapping types is then compared to the competing Adaptive Mapping algorithm and Traditional GP with respect to three regression benchmarks. PAM DGP using redundant mappings is then applied to two medical classification domains, where PAM DGP with redundant encodings is found to provide better classifier performance than the alternative algorithms. PAM DGP with redundant mappings is also given the task of learning three sequences of increasing recursion order given a function set consisting of *general* (not implicitly recursive) machine-language instructions; where it is found to more efficiently learn second and third order recursive Fibonacci functions than the related developmental systems and Traditional GP. PAM DGP using redundant encoding is also demonstrated to produce the semantically highest quality solutions for all three recursive functions considered (Factorial, second and third order Fibonacci). PAM DGP is shown for regression, medical classification, and recursive problems to have produced its solutions by evolving redundant mappings to emphasize appropriate members within relevant subsets of the problem's original function set.

Keywords: developmental genetic programming, genetic code, cooperative coevolution, genotype-phenotype mapping, redundant representation, neutrality, recursion

List of Abbreviations and Symbols Used

α	learning rate for PAM DGP
γ	noise threshold for PAM DGP
AE	Arficial Evolution
ADF	Automatically Defined Function
ARN	Artificial Regulatory Network
BNF	Backus-Naur Form
CCEA	Cooperative Coevolution Evolutionary Algorithm
CE	Computational Evolution
DGP	Developmental Genetic Programming
DNA	Deoxyribonucleic acid
EC	Evolutionary Computation
EIHC	Exhaustive Iterative Hill Climber
FIFO	First In First Out
GA	Genetic Algorithm
GCT	Genetic Code-like Transformation
GE	Grammatical Evolution
GP	Genetic Programming
GPM	Genotype-Phenotype Mapping
IDS	Intrusion Detection System
LGP	Linear Genetic Programming
LTFE	Lifetime Fitness Evaluation
MAX	Maximum Output Problem

ML	Machine Learning
MPS-CEA	Multipopulation Symmetric Cooperative Coevolutionary Algorithm
mRNA	Messenger Ribonucleic Acid
MSS	Mediocre Stable State
MTQ	Maximum of Two Quadratics
PAM DGP	Probabilistc Adaptive Mapping Developmental Genetic Programming
PC	Program Counter
RNA	Ribonucleic Acid
tRNA	Transfer Ribonucleic Acid
VRM	Virtual Register Machine
XO	Crossover
XO-EIHC	Crossover-Exhaustive Iterative Hill Climber hybrid
XOR	Exclusive OR (logical operator)

Acknowledgements

Firstly, I must thank my supervisor Dr. Malcolm Heywood. I am truly fortunate to have such an advisor, and I'm not sure I could ever repay him for all that he has done for me. His guidance and advice throughout my graduate career have been invaluable, as well as the opportunities he has provided me to publish and present our research at national and international venues. I would also like to thank my graduate committee members Dr. Nur Zincir-Heywood and Dr. Qigang Gao for their work and valuable feedback on my research aptitude exam, thesis proposal defense, and final thesis defense. I would also like to thank Dr. Wolfgang Banzhaf very much for sharing his ideas on the future and current direction of developmental genetic programming and for his efforts in proposing a potential research fellowship. A number of anonymous referees have contributed to this work through their feedback on submitted papers, and a number of their suggestions have been incorporated in this work. If you are one of those people and you happen to be reading this, thank you.

I would also like to gratefully acknowledge the financial support of an NSERC PGS-B, Izaak Walton Killam scholarship, and the Dalhousie Faculty of Graduate Studies and Faculty of Computer Science (Garnett Wilson) and CFI New Opportunities and NSERC Discovery research grants (Dr. M. Heywood).

On a personal level, I wish to thank Michelle Scaplen for her love, patience, encouragement, and support as I completed this thesis. I must be the luckiest man in the world.

Chapter 1. Introduction

1.1 Traditional Genetic Programming

Traditional genetic programming (GP) incorporates a group of machine learning techniques from the larger set of methods sharing a neo-Darwinian motivation known as "Evolutionary Computation" (EC). EC methodologies will be described by the way that they address three machine learning issues: representation of the problem and potential solutions, specification of the problem objective, and issues involving the way that the search for a solution is conducted. These elements of particular EC methodologies are explained in this section to the extent that they allow a brief background so that a reader who is unfamiliar with the EC field can understand the research presented in this thesis.

In terms of the problem representation issue, EC algorithms are search and optimization techniques that artificially replicate a neo-Darwinian concept of natural selection. As in actual biological models, a population of individuals is considered. In EC, the individuals in the population represent a set of candidate solutions to an optimization problem. In Genetic Programming (GP), as opposed to other EC methods, the structure of an individual is executable code, where the execution of the code determines the fitness of an individual. In the most traditional GP model, the executable code took the form of a tree structure as specified by Koza [51]. The instructions in each individual were comprised of zero-argument instructions (members of a Terminal Set) and instructions with one or more arguments (the Functional Set). In the tree structure implementation, the Functional Set members were present in the internal nodes of the tree and the Terminal Set members were at the leaves. This work uses a more modern variant of genetic programming called "linear genetic programming" or "LGP" [15]. In this

variant, the individuals have the form of a linear list of instructions rather than a tree-based structure. Program execution is that of a simple register machine (Von Neumann computer), and instructions are made up of opcodes and operands (providing linear forms of Functional and Terminal sets, respectively). Tree structural representation makes terminal and function set independent entities, however the linear model references both in every instruction. As the program executes, it alters the contents of internal registers or a stack and solution register. The structure of a linear GP individual is depicted below in Figure 1.1. The internal registers or stack provide a means of storing sub-results and can reduce the need to introduce new operators into a problem's Functional Set.[1] Linear GP is also more flexible than tree-structured GP, since each instruction in a linear GP individual does not necessarily contribute to the result in the solution register [15]. In contrast, in tree-structured GP, each node in the tree contributes to the final value found in the root node. Linear GP thus allows for more redundancy and for the presence of detrimental code in a solution without it affecting the fitness of an individual. An individual's program in linear GP consists of a string of bits. These bit strings constitute the individual's raw genetic material, or genotype. When the bit strings are interpreted, they correspond to members of the Functional (and sometimes Terminal) sets to produce a solution that makes semantic sense in terms of the original problem, also called the "phenotype." For instance, the binary sequence "011" in the individual's genotype could be interpreted as the functional set member "addition" in the phenotype. The phenotype is then evaluated to determine the corresponding fitness, bringing us to the second machine learning issue distinguishing GP algorithms from other methods.

[1] The terms "function" and "operator" are used in this discussion interchangeably, where "operator" is sometimes used to refer to functions in the linear GP literature.

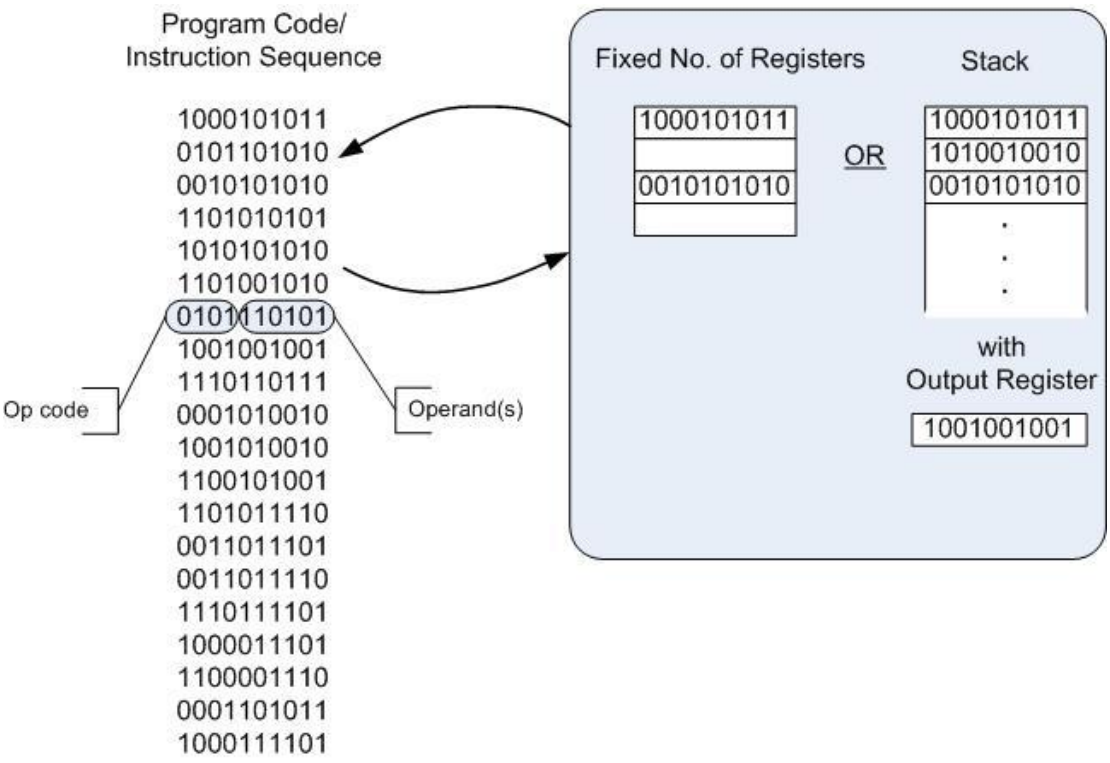


Figure 1.1. Structure of a Linear Genetic Programming (Linear GP) individual.

The second issue is the specification of the problem objective, and the associated measure for determining how well an individual meets the stated objective. The individuals in the population are ranked in their ability to perform the objective of solving the optimization problem based on some measure of error or success, called a "fitness" or "cost" function. The fitness functions are specified by the programmer *a priori* to reflect the nature of the problem. In linear GP, the execution of the program determines an individual's fitness. Unlike other machine learning methods, GP provides a lot of flexibility in the form of the fitness function. That is to say, the user is typically free to specify a fitness function that directly reflects the goals of the problem domain, without having to incorporate constraints from the machine-learning model. For example, smoothness constraints are required for kernel and neural network models. A natural penalty for this freedom, however, might be a higher computational cost associated with

the training phase, where this may or may not be outweighed by providing a more appropriate solution (on account of more clearly defined goals).

The third and final issue in machine learning is the way in which the search for a solution is conducted. The individual candidate solutions in the population are ranked using the fitness function, and the fittest individuals are biased for selection as parents and used to create a new population of solutions. The genetic material composing the parents is then manipulated (sometimes after being copied) to create children who typically replace some individuals originally in the population. Population size thus does not change. The manipulation of genetic material is accomplished through the use of the two operators of crossover and mutation in this work (and most others). Crossover swaps the genetic material of the children (parent copies), thus exploiting genetic material already available. Numerous types of crossover have been devised [15], but in this research two instruction segments of equal size from each of the children individuals, chosen with a uniform random probability, are exchanged if a particular probability threshold is exceeded. The size of individuals is used as a constraint in the first problem considered in this thesis, and this type of crossover will always allow individuals to remain the same length. The crossover operator is demonstrated in Figure 1.2 below.

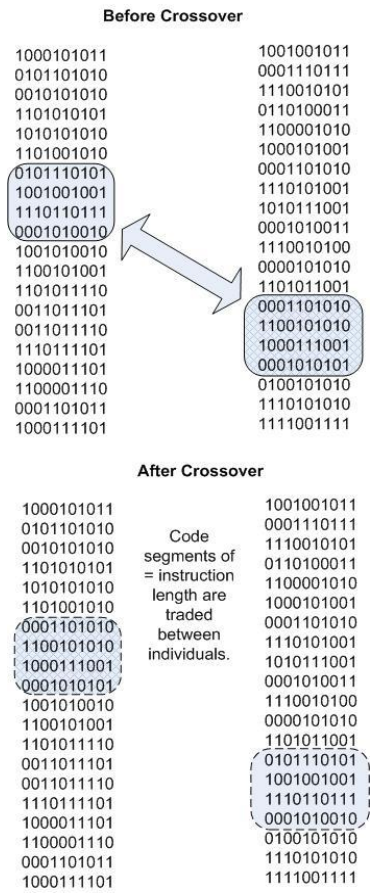


Figure 1.2. **Crossover of equal-sized instruction sequences.**

Mutation changes the genetic material already present in a child, providing new genetic material to be explored. In this research, two types of mutation are used: The first type is called "point mutation," where each bit in a genotype string is processed sequentially and changed from a 0 to a 1, or vice versa, if a particular threshold is exceeded. The other option for mutation is XOR mutation, where a bit string comprising an instruction is chosen with a uniform random distribution if a threshold is exceeded. The bits in that instruction then have a randomly generated bit mask applied to them, and the XOR operator is applied to each pairing of bits at each position in the instruction and the mask to generate a new bit sequence. This new bit sequence then replaces the

instruction originally chosen for modification. As such, this form of mutation tends to modify the entire instruction as opposed to a single field within an instruction. Both procedures are given in Figure 1.3 below. The basic algorithm for a GP incorporating these operators, as described in [51], is shown below in Figure 1.4.

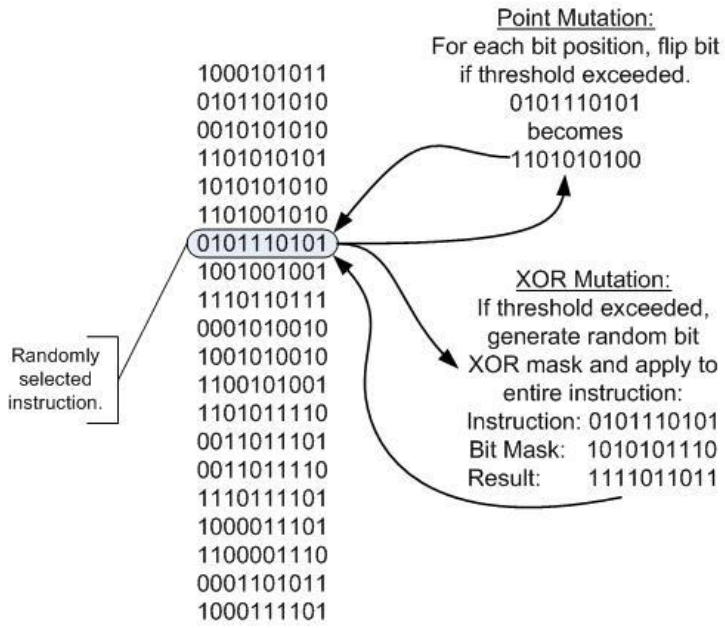


Figure 1.3. Point mutation and XOR mutation.

```
Specify instruction set.
Stochastically[2] generate initial population of individuals.
while (tournament not done && desired fitness not found)
   Execute program comprising each individual to determine
      fitness
   Select two parent individuals
   Apply crossover to parents' genetic material to create
      children with a priori probability
   Apply mutation operator to children with a priori
      probability
```

Figure 1.4. Pseudocode for the general Genetic Programming (GP) algorithm.

[2] This typically takes the form of a series of stochastic decisions, selecting between different instruction types, and operand or opcode values depending on the instruction type [15].

The general GP procedure just given can be specialized into two forms of tournament selection: generational and steady state. In the generational approach, the fitness of an entire population is evaluated. Every individual in the population is then ranked, and it is assigned a normalized probability of selection based on its fitness. (The probability of selection of a given individual is its fitness divided by the fitness of the total population.) Pairs of individuals are then iteratively selected, and tests for applying crossover and mutation are applied to them. The iterative selection continues until there are sufficient children present to represent an entirely new population. The operation of the generational tournament approach is shown below in Figure 1.5.

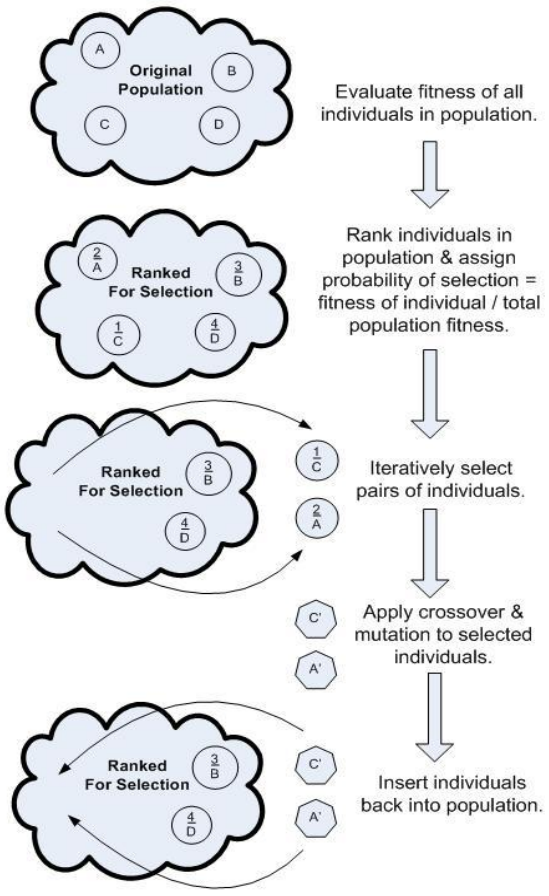


Figure 1.5. Generational tournament selection.

The alternative to generational tournament selection is steady state tournament selection. It relies more heavily on random selection than the generational tournament, and is the choice of the algorithm that is the main subject of this thesis, and its most modern predecessor [57-59]. In this type of selection mechanism, a small subset of individuals is randomly selected from the population to take part in a tournament round. The fitness of the chosen individuals is evaluated, and the fittest half of the individuals is selected as parents. Individuals from the less fit half of the tournament become children, with their genotypes overwritten by those of the parents. Mutation and crossover are then stochastically applied to the children to recombine existing genetic material and introduce new material, respectively. The children then overwrite the corresponding number of worst fitness individuals in the population or in that tournament round. The steady state tournament approach is shown below in Figure 1.6.

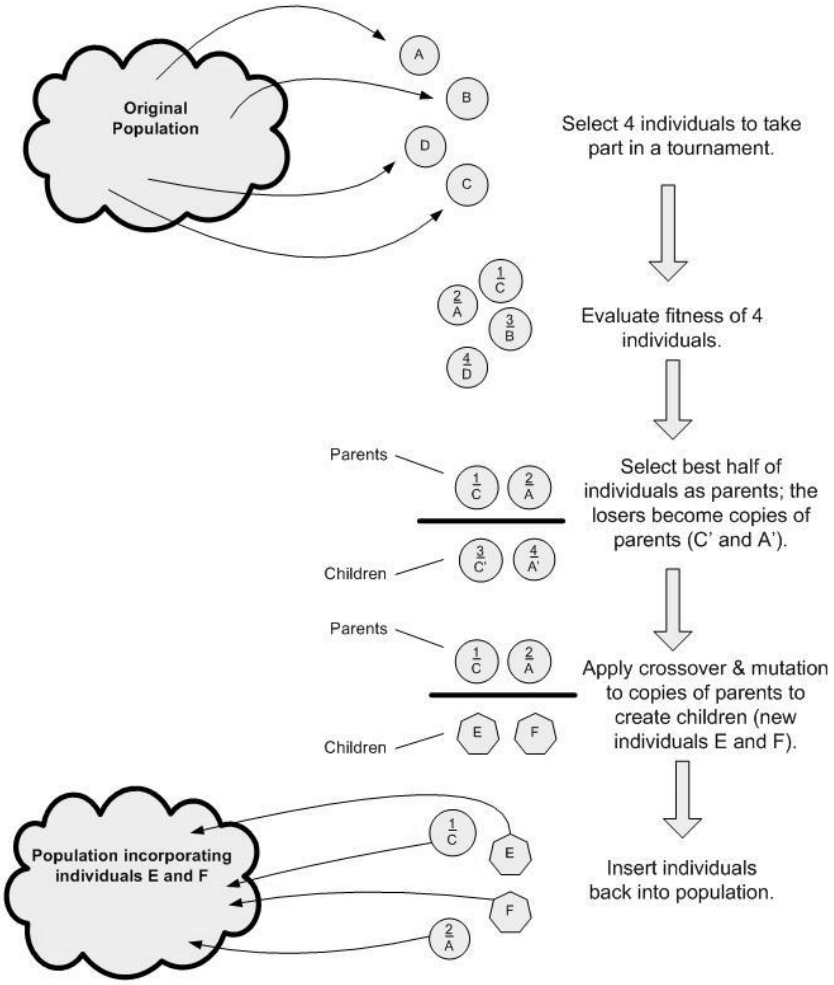


Figure 1.6. Steady state tournament selection.

1.2 Separation of Genotype and Phenotype

Given the explanation and background of GP above, it is evident that traditional genetic programming does not discern between a genotype and its corresponding phenotype in the search space. Broadly speaking, the term "developmental genetic programming" (DGP) includes methodologies that explicitly set out to separate the genotype space from the phenotype (or solution) space through a connection (or mapping) between the two spaces [45]. In the literature, the term has been used to encompass systems, such as the one presented in this work, that insert a genotype-

phenotype mapping (GPM) as a genetic code to establish a relationship between the two spaces. A genetic code in the context of this work follows the definition provided by Keller and Banzhaf in [45], that is, it is the encoding of a symbol by one or more codons where a codon is a non-zero contiguous bit sequence from a binary genotype. In terms of describing the developmental context (the biological analogue) of the constructs in the algorithm produced in this thesis, the system can be considered to model the evolution of individuals and a shared "genetic code" that maps codons to amino acids in biological organisms.

This work conducts an investigation of the algorithmic framework used to coevolve populations of genetic code mappings and genotypes, as well as identifying the most appropriate GPM model for use in such a coevolutionary framework. In the process of meeting these requirements, a new algorithm for the coevolution of efficient mappings, called Probabilistic Adaptive Mapping Developmental Genetic Programming (PAM DGP), is presented. A preliminary version of PAM DGP was first presented in [95], with considerable improvements introduced in [96] (wherein the performance of PAM DGP was also demonstrated on two regression problems). [94] served as the comprehensive introduction of PAM DGP to the literature, significantly expanding on [96] by introducing a superior (and more developmental) adaptive redundant mapping structure to the algorithm, comparing PAM DGP to other GP algorithms using medical classification benchmarks, and providing an extensive discussion of the developmental nature of PAM DGP and how it addresses coevolutionary pathologies in the previous work of Margetts and Jones [57-59].

The system described in this work has its roots in the DGP algorithm of Keller and Banzhaf [5, 45-47], but separates mapping from genotype (they are united in the implementation of Keller and Banzhaf). Mappings and genotypes are separated into two populations that coevolve as in the subsequent algorithm of Margetts and Jones [57-59], called the Standard Adaptive Mapping DGP (or simply Adaptive Mapping DGP) in this thesis. The system implemented here uses a mapping that can be seen as a table relating genotype segments (binary sequence codons) to symbol members of a function set, and in that respect it is similar to the mapping structure that serves as an analogue of the biological genetic code seen in the implementations produced by both groups of researchers. Their collective work thus serves as the starting context for this work.

The evolution of a genetic code is of great benefit when there is little or no information about the problem space, or when the problem space takes the form of a dynamic environment and adaptiveness is required for survival of individuals. An evolved genetic code, or mapping, is used to adaptively emphasize symbols that are important for the solution and dictate which symbols are permitted for use in the construction of a solution. The co-evolution of the mappings and genotypes thus dynamically reduces problem search space and biases solutions towards particular regions of the search space. Furthermore, as noted by Keller and Banzhaf [47], the separation of the two spaces avoids the hindering effect of operators in traditional GP approaches that are restricted so that only legal phenotypes are generated. This built-in restriction of the operators limits the search to the feasible areas of the search space. However, the infeasible regions may contain genetic diversity needed to quickly generate very high quality individuals within fewer generations. The aim of this work is to

provide a system that efficiently generates mappings and their corresponding genotypes to realize the benefits of both a more efficient search and a better tailored solution that incorporates and emphasizes the correct symbols in a problem's function set.

1.3 Objectives of Research

There are three other systems that evolve genetic code-type mappings and solutions in a developmental GP approach. Keller and Banzhaf [5, 45-47], Margetts and Jones [57-59], and O'Neill and Ryan [67, 69] all consider their approaches developmental systems that evolve a mapping that is an analogue of the biological genetic code. Margetts and Jones [57-59] and O'Neill and Ryan [67, 69] both consider their systems to implement coevolution. Keller and Banzhaf [45, 46] do not explicitly state that they implement coevolution, although their method of evolving both mapping and genotype solution (as paired individuals) is identical to O'Neill and Ryan. Keller and Banzhaf are credited with using coevolution by O'Neill and Ryan in [69], where they state "studies [Keller and Banzhaf] provide strong evidence demonstrating the effective co-evolution of genetic code and solution." In any case, all research teams implement developmental systems like PAM DGP where genetic codes are evolved along with genotype solutions. We make a significant contribution to the area of genetic code-based models of developmental GP, where the objectives of this thesis are to:

- Identify, and empirically demonstrate, serious performance problems caused by coevolutionary pathologies and lack of exploration of the search space for the latest (Standard) Adaptive Mapping DGP algorithm.

- Introduce a robust new algorithm, Probabilistic Adaptive Mapping Developmental Genetic Programming, or PAM DGP, and show that it overcomes coevolutionary pathologies without incurring significant computational cost and avoiding premature convergence on local optima. (To the author's knowledge, this is the first instance of a developmental GP algorithm based on evolution of genetic codes that was designed to avoid coevolutionary pathologies. The overall means with which the algorithm avoids the pathologies is also novel.)
- Establish empirically that PAM DGP provides a superior algorithmic framework, given equivalent genotype and mapping structures as the Standard Adaptive Mapping algorithm, on three regression benchmarks (MAX Problem, Hénon map, and Two Boxes).
- Introduce a new adaptive redundant mapping for the PAM DGP algorithm for superior fitness-based performance on harder problems and closer adherence to developmental modeling of the biological code.
- Empirically examine the performance of PAM DGP with adaptive redundant mappings on previous regression benchmarks.
- Demonstrate the ability of PAM DGP with redundant mappings to generate higher accuracy classifier systems on medical classification benchmarks from the UCI repository than PAM DGP with alternate (less developmental) mappings, the Standard Adaptive Mapping algorithm, or Traditional GP.
- Show that PAM DGP with redundant mappings learns recursive solutions given a function set consisting of *general* (not implicitly recursive) machine-language

instructions for three recursive sequences of increasing order (factorial, Fibonacci, and third order Fibonacci).

- Demonstrate the PAM DGP provides the semantically highest quality solutions for every recursive function (factorial, Fibonacci, and third order Fibonacci) over all algorithms considered in this work.
- Demonstrate that the redundant mapping PAM DGP algorithm produced its solutions to the regression, classification, and recursive functions through evolution of redundant mappings that effectively selected, and emphasized members of, particular subsets of the function set for higher fitness-based performance.
- Showcase the robustness and customizability of the PAM DGP framework throughout the thesis using the chosen benchmarks, while making practical recommendations for parameterization of the PAM DGP algorithm when solving problems of various types and difficulties.

1.4 Thesis Overview

Chapter 2 of this work describes background literature relevant to developmental systems and coevolution of genetic code-type mappings and solutions. In particular, the chapter describes the developmental nature of PAM DGP, briefly reviews other current avenues in developmental systems, examines proposed genotype-phenotype mapping models, and describes types of coevolution and their pathologies. The chapter also reviews alternative mappings used in other coevolutionary implementations, with particular focus on the works of Keller and Banzhaf [5, 45-47] and Margetts and Jones [57-59] in which this work has its roots.

Chapter 3 details the drawbacks and coevolutionary issues of the earlier Standard Adaptive Mapping DGP algorithm of Margetts and Jones [57-59], proceeding to introduce the PAM DGP algorithm. The chapter then examines how PAM DGP improves on the Standard Adaptive Mapping DGP and the computational complexity of the two algorithms is then discussed. Chapter 4 compares the performance of the two algorithms on the simple regression Maximum Output (MAX) problem previously used to introduce the Standard Adaptive Mapping DGP to the literature [58], with Chapter 5 comparing the two algorithms on the harder Hénon mapping and Two Boxes regression problems.

Having established performance comparisons of PAM DGP and Margetts and Jones's algorithm using equivalent genotypes and mappings, a more developmental adaptive redundant mapping is incorporated in PAM DGP in Chapter 6. Chapter 6 also discusses the expected benefits of introducing an adaptive redundant mapping and briefly reviews literature on redundant representations in Evolutionary Computation and the theory of adaptation through neutral variation. Computational complexity benefits for PAM DGP resulting from the introduction of the new mapping encoding is provided, and the chapter finishes with an analysis of the MAX problem using the new mapping.

Chapter 7 compares PAM DGP (using both mapping types), Margetts and Jones's Standard Adaptive Mapping algorithm, and Traditional GP on two medical classification benchmarks. The analysis in Chapter 7 demonstrates the effectiveness of the new adaptive redundant mapping in PAM DGP with respect to accuracy of the classifiers, and shows it is achieved through tailoring of the function set. Chapter 8 applies PAM DGP with redundant encodings to learning recursive solutions given a function set consisting

of *general* (not implicitly recursive) machine-language instructions. PAM DGP using redundant encodings is found to more efficiently learn second and third order recursive Fibonacci functions than the related developmental systems examined in this thesis and Traditional GP. PAM DGP using redundant encoding is also demonstrated to produce the semantically highest quality solutions for all three recursive functions considered (Factorial, 2nd and 3rd order Fibonacci). PAM DGP is then shown to have produced such solutions by evolving redundant mappings to select and emphasize appropriate subsets of the function set useful for producing the naturally recursive solutions.

Chapter 8 provides discussion, including guidance for readers on setting appropriate parameters in PAM DGP and limitations of the algorithm. Summary and conclusions follow, finishing with some directions for future work.

Chapter 2. Literature Survey and Background

This chapter reviews previous work relevant to the contributions of this thesis, where the research presented here covers a number of research areas within the broader field of Evolutionary Computation. These areas are covered in Section 2.1 of this work, where previous work in fields of developmental systems, genotype-phenotype mappings (GPMs), coevolution and its pathologies, and coevolved mappings are discussed. Sections 2.2 and 2.3 discuss in greater detail the two genetic code-based coevolutionary systems that are the most relevant background to this thesis. We begin with a description of the biological nature of the developmental model used in this work.

2.1 Context of this Work in the Current Literature

2.1.1 The Developmental Nature of PAM DGP

The biological process of decoding genes to produce proteins, called "protein synthesis," occurs in two stages: transcription and translation. We provide here a very simple account of aspects of protein synthesis relevant to this thesis: During transcription, sections of the double-helix deoxyribonucleic acid (DNA) are unwound to expose the genes. The genes are then used as templates to create messenger copies of the genes (mRNAs) in the DNA's analogue chemical language ribonucleic acid (RNA). During the translation phase, the codons (triplets of nucleotides) on the mRNA molecules are recognized by one end (called the "anticodon") of a transfer RNA (tRNA) molecule. The other end of the tRNA molecule corresponds to a binding for a specific amino acid.

The correct tRNA molecules continue to bind to successive codons, bringing together the required amino acids in a "polypeptide chain" to form the protein product for which the gene was originally coded. Multiple codons of the mRNA (and hence, multiple genes of the DNA) can specify the same amino acid. However, each type of tRNA molecule can be attached to only one type of amino acid. Thus, multiple types of tRNA molecules exist with identical anticodons that can carry the same amino acid. All this amounts to a redundant code: each amino acid is specified by more than one codon [55]. The description of protein synthesis is depicted below in Figure 2.1.

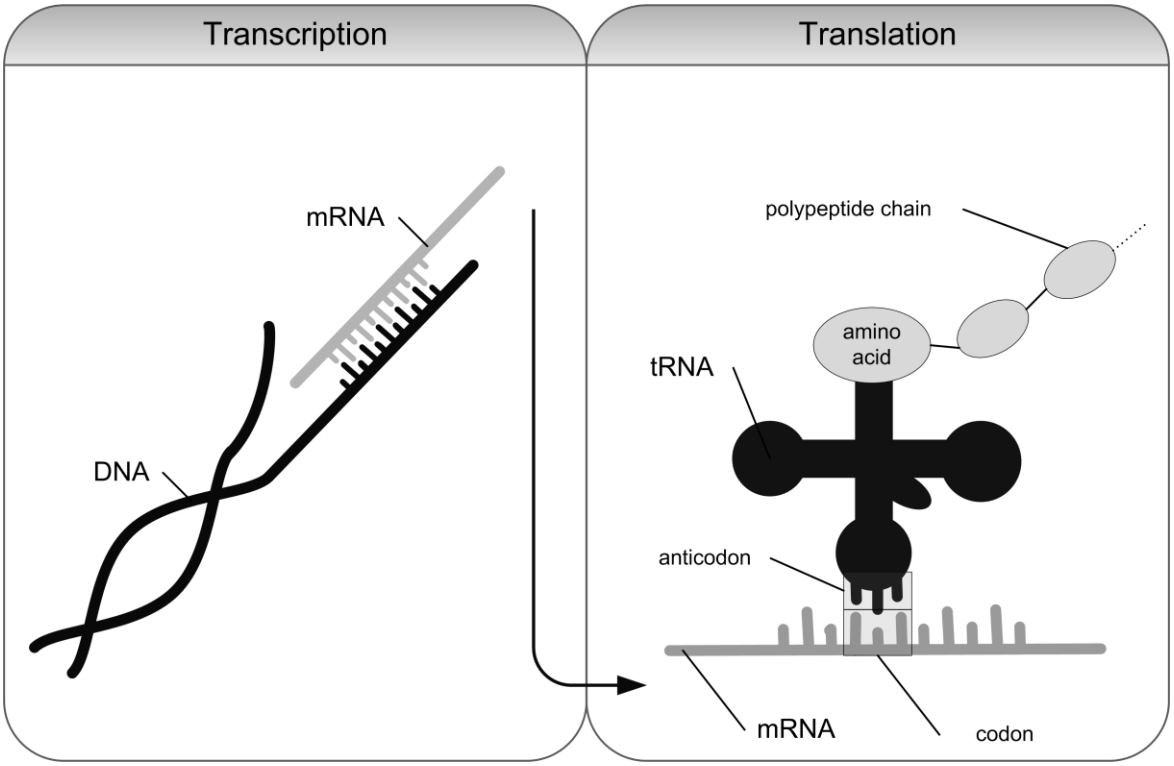


Figure 2.1. Aspects of protein synthesis relevant to genetic code-based DGP.

The biological development equivalent of a genotype-phenotype mapping individual in our DGP is technically the tRNA molecules—they are responsible for

mapping codon (via anticodon identifier on the tRNA) to amino acid. (To the author's knowledge, O'Neill and Ryan originally draw the analogy of modeling the tRNA molecule when using a genetic code-based GPM in [69].) Another, simpler way of putting this is that the genetic code (biological codon to amino acid mapping) is being modeled. Since the natural genetic code is redundant, a developmentally accurate DGP analogue ought to be redundant. Regarding research groups to be discussed in this section, Banzhaf and Keller [5, 45-47] and O'Neill and Ryan [67, 69] incorporate redundancy into their mappings, whereas the encoding of Margetts and Jones [57-59] is one-to-one (and hence neither redundant nor developmentally accurate).

The purpose of the coevolutionary developmental system described in this thesis is to evolve both a good genetic code and an optimized solution. But the genetic code is often assumed to by the layperson (and modeled in most traditional evolutionary algorithm systems) to be fixed. As it turns out, modern biology indicates that the genetic code is *adaptive*: it is capable of evolving [28, 90]. Furthermore, the genetic code itself has evolved as a product of adaptive evolution [28], and its evolution is capable of explaining several problems in evolutionary biology [90]. In particular, it has been postulated that the genetic code adapts to minimize the negative effects of mutation and gene translation errors while also maximizing the rate of natural selection as a search algorithm [28]. Recent work in the field of molecular evolution argues that the particular mapping of codons to amino acids dictated by the current genetic code is a product of the continued coevolution of the genome and the genetic code [83]. Crick's widely-accepted theory actually always held that coevolution occurred between the genetic code and the genome, but that the coevolution ceased once the current associations between codons

and amino acids were established in the primitive genetic code. Crick's theory [21] also held that as evolution of the primitive genetic code proceeded, the number of proteins in the genome increased and their design became more complex. Crick believed that when the genetic code reached its current form, it was so highly developed that any change in it would have introduced new amino acids into the highly adapted proteins and spelt disaster for the organism. The genetic code thus became a "frozen accident." Crick himself was critical of his own theory, finding it too accommodating. Sella and Ardell [83] have since found that coevolution modeled with errors in replication and translation actually generate the codon-amino acid associations rather than simply preserving ancestral versions of the associations as Crick supposed. In addition, the models of Sella and Ardell consistently generated the main organizational features of the standard genetic code. The adaptive nature of the genetic code in biology justifies the application of selection to a population of genetic codes in a developmental system. The fact that the mapping of the genetic code is suspected to co-evolve with the genome justifies the use of coevolution in the developmental model.

It is still an open question how best to implement this coevolution. The mappings (tRNA molecules) and genotypes (genes) are separate molecular entities within a single molecule, so it may seem to make sense to evolve them as being paired within an individual (as in [45-47]). However, if we consider this a bit further, such an approach will not accurately model biological entities as a population. At any given time in evolutionary history, each separate individual will possess unique DNA but undergo basically the same transcription / translation processes (with very rare exceptions that slowly forward the evolution of the genetic code). That is, each individual does not

possess a unique genetic coding scheme that occurs within their own cells. The aim of the developmental model is to determine how the genetic code has evolved to optimize for the *collective* population of genotypes. In nature, a highly evolved and nearly universal standard genetic code developed very early in the history of life whereas our human genome is a much newer development. As noted in by Freeland [28], natural selection processes approach optima asymptotically. As the genetic code is now highly adapted, the changes occurring to it are practically negligible. However, problems posed to an artificial DGP system cannot assume to have been handed a near optimal genetic code as a context in which to work. The purpose of the coevolution of the genetic code and genotypes in DGP systems is to play "catch-up"—evolve an optimized (or as optimized as possible) genetic code in the context of the *population* of genotypes rather than an individual's genotype. It would thus seem more appropriate to evolve candidate genetic codes (mappings) against a collective population of genotypes in a separate coevolutionary population. Researchers who have opted for this approach include [57, 58] and the authors.

2.1.2 Other Developmental Research

Other developmental approaches also include systems that involve types of constructs other than the genetic code between the genotype and phenotype spaces. This Section very briefly covers such alternative approaches, and is included to give the reader a taste of current research and the direction of the exciting and expanding field of developmental and generative systems. Detailed discussion of developmental systems

incorporating GPMs directly relevant to the algorithm introduced in this thesis will be provided in the upcoming Sections.

Some GPMs in developmental systems incorporate a complex of relationships. One such system proposed by Banzhaf [4, 6, 7], called Artificial Regulatory Networks (ARNs), relates the genotype and phenotype spaces by mimicking the process of protein production in nature. The genotype is used to encode information (located at a "promoter" site) regarding the amount of genotype to be read to produce proteins. The proteins that are produced interact with either the genotype or other proteins. The occupation of particular sites on the genotype (promoter or inhibitor sites) upstream from the promoter site by proteins dictates the efficiency of the expression of the gene downstream from those sites. The system results in a regulatory network with genes as nodes and proteins creating links between the nodes. The weight of the linkage between nodes is determined by the degree to which a protein and regulatory site on the genotype match. Another well-known approach by Koza is to separate a genotype from a phenotype expressed as a circuit [50, 52]. An initial embryo consists of an unconnected wire, and it is developed into a circuit through the application of the functions in evolved tree-based genotypes. Thus, the genotype serves as a design plan that is applied to the initial embryo wire to form a more complex circuit.

The current research in the developmental GP field is so rich and varied that algorithms and associated structures are often unique to individual research teams and their associated projects. Unique developmental models have been applied to the design of robots and robot controllers by prominent researchers in the field such as Bentley, Hornby, and Bongard [10, 13, 14, 35, 36]. Developmental systems have also been used

for the automatic design of circuits and hardware (such as antennae) by researchers including Lohn (with Hornby), Koza, and Gordon (with Bentley) [32, 50, 56], as well as used to provide self-repairing capabilities and graceful degradation to systems by Miller and Banzhaf, Miller and Tyrrell, and Bentley [11, 60, 61]. Lists of researchers and references associated with each area just mentioned are not meant to be exhaustive, but provide the reader with a starting point for current research applications of developmental GP as of this writing.

The future of the developmental GP field seems to be headed toward a close adherence to evolutionary phenomena as they are understood from nature, and it is poised to impact the broader evolutionary computation field. Since Banzhaf and Keller authored the pioneering work in which this thesis has its roots [5, 45-47], Banzhaf has formulated suggestions for the most promising direction for the future of the field. A number of prominent researchers in evolutionary computation and biology, including Banzhaf, have recently called for a research program with the aim of creating a new field called "computational evolution (CE)" [8]. The new field will involve the creation of new algorithms to modernize current evolution-inspired implementations so that they more closely reflect a contemporary understanding of natural evolution. The current implementations based on simplified models of biological evolution are dubbed "artificial evolution (AE)" by Banzhaf *et al.*, and are specified to include "evolutionary programming, genetic algorithms, evolutionary strategies, and genetic programming" [8]. In these implementations, the genomes (genetic content) of individuals are typically small and are directly mapped to simplified phenotypes (expressions of the genetic material). AE algorithms seldom use more sophisticated mechanisms such as self-modification or

feedback that are seen throughout nature, and AE algorithms typically proceed to a static objective that is defined *a priori*. In contrast, natural evolution is now known to operate at numerous levels of complexity. DNA, the genetic code, and control and feedback mechanisms are themselves now understood to be products of evolution; there is not a unidirectional flow of information from DNA to protein. It has also long been known that there is differential gene expression and more than one genotype can map to more than one phenotype in nature. Furthermore, evolution in nature is open-ended and fosters emergent phenomena such as re-use of genotypic and phenotypic structures for novel purposes, regulatory networks, dynamic mappings, and formation of new species.

The hope of CE is that its algorithms, by a closer adherence to the intricacies of natural evolution, will be powerful tools for solving complex computational problems that have proven difficult for AE approaches. Such problems include such features as: not being completely definable *a priori*, dependence on temporal variations, difficulty discerning relevant data from information available for the solution, and the requirement that a solution must be determined completely autonomously. Examples of such problems for which CE could contribute cutting edge solutions include environmental sensing in robots that alter their own environment, data mining of complex dynamic data sets, innovative design (special purpose programs and hardware), autonomic software systems (systems of interacting software objects that collectively work toward a globally defined goal, while being able to react gracefully to perturbations such as damage and increased demand), problems with unknown or overly complex specification issues, and other open-ended (yet constrained) problems that could not previously be tackled. It is

hoped that this work will prove to be a valued step in the direction of establishing this new paradigm.

2.1.3 Genotype-Phenotype Mappings

Any attempt to separate genotype and phenotype space, including the developmental systems discussed in the previous two sections, involve design decisions regarding the definition of the mapping between the two spaces. Broadly construed, the literature has used the term "genotype-phenotype mapping" (GPM) to indicate any means at all of allowing genotypes to translate to phenotypes. Bentley and Kumar have endeavored to produce a classification of genotype-phenotype maps [12]. To be more specific, Bentley and Kumar prefer to consider mappings as growth processes, and alternately call them "embryogenies." The first type of embryogeny is the *external (non-evolved)* type, which is user-designed, defined globally, and is external to genotypes. Such embryogenies are static during the evolutionary process.

The second type of embryogeny is *explicit* (*evolved*). This type is said to be a data structure or set of instructions that dictate how the genotype is to be interpreted to construct the phenotype. Bentley and Kumar state that such systems typically combine the genotype and mapping and allow the evolution of both simultaneously. While the work of Keller and Banzhaf [5, 45-47] combine genotype and mapping in a single individual, coevolutionary systems use one population of genotypes and one population of mappings as separate individuals. This is the approach taken in this work, as well as in the implementations of Margetts and Jones [57-59] and O'Neill and Ryan [67-69].

The third, and final, type of mapping in Bentley and Kumar's taxonomy is the *implicit* (*evolved*) embryogeny. In this type of mapping, an indirect series of potentially interacting rules is used to generate the phenotype from the genotype. These rules are not typically executed in a predetermined or preprogrammed order, but can be used in a dynamic, parallel, and adaptive way. Examples of implementations of this type include Banzhaf's Artificial Regulatory Networks (ARNs) [6, 7], Koza's evolved circuits [50, 52], and a number of evolvable hardware approaches such as those mentioned in Section 2.1.2. The term *mapping* is used in this thesis to refer to a correspondence of one set (genotype codons) to another (function set members), rather than complex networks or processes to relate the genotype and phenotype spaces. A mapping, so construed, is of the *explicit* (*evolved*) embryogeny variety. Furthermore, such mappings that relate codons to function set members (assuming they are properly constructed) can also be used in more complicated growth structures. That is, well-constructed explicit embryogenies (if a correspondence of one set to another) can be incorporated into larger implicit embryogenies. This would be analagous to the genotype-phenotype mapping mechanism (tRNA) being incorporated in the protein synthesis system of a cell.

A number of types of GPM alternatives exist, even when only construing mappings as a correspondence to relate genotype codons to traits of the phenotype (as opposed to complicated or dynamic developmental processes). A simple external (globally applied) mapping example would be that of Bean [9], where he was attempting to provide solutions to a job shop scheduling problem such that each genotype represented a valid solution when interpreted as a phenotype. His idea was to create a mapping that sorted the alleles (gene values) and sequence the jobs (one corresponding to

each allele) in ascending order of the sort. The values given to the alleles were randomly chosen real probabilities in the interval [0, 1], and his representation technique was called "random keys." His implementation was thus simply a sorting mechanism of randomly generated real numbers, amounting to a random search in the mapping space.

Altenberg [1, 2] uses an external, globally defined genotype-phenotype mapping that is adapted with each new gene created in the genotype space. Since the mapping is externally defined, but adaptive (not static) it does not appear to fit into Bentley and Kumar's taxonomy, being somewhere between external and explicit. In Altenberg's representation, a genome consists of a number of genes with binary values that determine a number of phenotypic functions, which in turn contribute to a component of the total fitness. Thus, each gene can have an effect on more than one function. The mapping is a matrix, each element of which is a binary value indicating whether or not each gene (column index) affects a fitness component (row index). In biology, polygeny is the phenomenon of one phenotypic trait being affected by more than one gene, while pleiotropy is one gene affecting more than one trait. The matrix thus provides information analogous to those two biological mechanisms: The columns of the matrix give vectors of the genes controlling each fitness value (polygeny) and the rows provide the fitness components controlled by each gene (pleiotropy). The motivation of his work was to demonstrate that the evolved mappings, given an appropriate selection operator for the inclusion of fitness-producing genes, kept genes that affected less fitness components (exhibited less pleiotropy) and created a smoother fitness landscape than an alternate approach.

Like Altenberg, Ohnishi [70] used a globally defined mapping that evolved during the course of the algorithm. Ohnishi's aim was to adapt a genotype-phenotype mapping so that a neighborhood in the genotype space is mapped to a number of separate areas in the phenotype space around local and global optima. In his implementation, the parameters of the mapping mechanism used to generate the phenotypes are part of the genotype space. These parameters are used in a time-sequential process when the genotype is interpreted, and mimic biological development in that respect. That is, the expression of the genes at a given time is affected by the expression of the genes that preceded them. The mapping maps a neighborhood in the genotype space into separate areas of the phenotype space by hierarchically clustering decision variables and assigning labels corresponding to ranges of final values to the clusters. Ohnishi shows the viability of his approach using two very simple test functions. In summary, Ohnishi uses a globally defined mapping with adaptable parameters that evolve throughout the algorithm because they are in the genotype space.

Kargupta and Ghosh [42, 43] introduced a mapping called a genetic code-like transformation (GCT) that is an analogy of the mapping of an mRNA sequence to a protein sequence. A number of 3-bit mRNA codons map to a single-bit protein feature. Thus, the coding is redundant because a particular protein feature can be produced by more than one mRNA codon. Reversing the usual order of translation, Kargupta's algorithm considers a protein sequence and transforms it into an mRNA space by assigning a codon sequence with uniform probability from the group that represents each protein bit. The implementation was used to construct a representation of a non-linear problem (the XOR function) with an error surface that is easy to minimize so that the

representation is learnable by a linear classifier (a Perceptron). Rather than attempting to refine or investigate mappings themselves, Kargupta and Ghosh are actually modeling a mapping in reverse to create a suitable training set in a binary problem domain.

The type of mapping that this thesis investigates is the type that can be construed as a "code book" or "genetic code," where the mapping is a direct relation that indicates a codon is associated with a symbol in the phenotype space. The only approaches discussed so far that treat a GPM explicitly as a genetic code in that traditional way are Keller and Banzhaf [5, 45-47] and Margetts and Jones [57-59]. However, another approach that treats a GPM as a genetic code in a less traditional sense, called Grammatical Evolution (GE), has been introduced by O'Neill and Ryan [68]. Their GPM maps a codon to a symbol in an output language by choosing a production rule corresponding to the current nonterminal in a Backus-Naur Form (BNF) grammar. A BNF grammar can be construed as the tuple $\{N, T, P, S\}$ where N is a set of nonterminals, T is a set of terminals, P is a set of production rules that maps N to T, and S is the start symbol (a member of N). Binary string genomes are used with codons of 8 bits. The integer values corresponding to the codons map to numbered production rules from the set P based on the mapping function (*codon integer value*) *modulo* (*number of rules for current nonterminal*). The interpretation of an individual's genome continues to loop over the genome until the mapping process produces an expression consisting entirely of terminals or the individual is determined to be invalid. Since many codons interpreted as integers can represent the same production rule, the mapping is redundant (more than one genome can represent the same phenotype). The mapping mechanism in GE is notably different than most other mapping studies, and represents a novel

contribution to that field that allows any computer language expressible in BNF form to be a GE solution.

2.1.4 Coevolution: Models and Issues

A number of GPM implementations, including this work and those to be discussed in the following section, employ coevolution. Research on coevolution is a broad and increasingly active area of research in its own right; for the purposes of this thesis we provide a brief introduction here with particular attention paid to coevolutionary pathologies and attempts to address them. Coevolution is a process where a given individual is evaluated using interactions with other individuals that are evolving at the same time [22], and it is usually divided into two types: cooperative and competitive. In the cooperative model, formulated by Potter and De Jong [78, 79], two or more populations are evolved where individuals only mate within their own population. The motivation behind cooperative coevolution is to decompose a complex problem into components, evolve the components in separate populations, and then assemble the components into a total solution. Since the populations are separate, individuals can (but need not) have very different structures so long as they can collaborate. To evaluate a member of one population, collaborations are formed with members of the other populations. The best individual from each of the other populations was chosen as the collaborator in the initial work on cooperative coevolution, and a phenotype was formed by combining the chosen individuals from each population [80, 81]. In the competitive model [22] originally introduced by de Jong and Pollack, individuals (called *learners*) are evaluated by being tested against other individuals (called *evaluators*). Learner and

evaluator are roles that individuals adopt during the course of an algorithm, and a particular individual can take on both roles in the course of the algorithm. The evaluator and learner can be individuals of similar type and evaluate one another in a two player game (in this instance, there is no need for separate populations of different individual types). However, separate populations of different types of individuals may be used in competitive coevolution, for instance where the evaluator is a test case (training exemplar) for the learner instance.

There are a number of problems that have been identified for coevolution in general. One problem is disengagement, where the performance levels for one or more objectives is at too high or too low a threshold in individuals of one population for another population to effectively collaborate with them to progress toward a solution [22]. This problem causes loss of gradient [91], where one population of individuals can become quite good at a particular objective but the individuals who interact with them cannot effectively learn from their achievements. (An example of this would be a beginner being unable to learn from a chess master.) Disengagement also typically leads to "over-specialization" or "focusing" [91], where individuals progress toward some problem objectives but not others. Another problem, specific to competitive coevolution among three or more populations, is intransitivity [22, 91]. In this situation, the preference measure between learners is not transitive: individual B appears preferable to A, C appears preferable to B, but A appears preferable to C. (Since the PAM DGP system described in this work is a two population cooperative coevolution implementation, intransitivity is not applicable but is mentioned for completeness of the survey of coevolutionary pathologies.)

A problem of concern in this work is referred to as the "Red Queen Effect" [20], named for the Red Queen character in Lewis Carroll's *Through the Looking Glass* who ran perpetually without moving because the landscape kept up with her. The Red Queen effect occurs in coevolution where traits in one population evolve against traits in the other population that are also evolving, thus little or no progress is made. What is hoped for in a coevolutionary implementation is an arms race, where progress is made by "mutual and reciprocal adaptations between collaborating groups of individuals" [93]. That is, what is needed is for each population to alternately build on the adaptations achieved by the other in order to mutually progress toward a solution. This situation is also known as a *parallel adaptive gradient* [93]. If some semblance of an arms race does not occur, the alternative is little to no progress toward a solution, where the adaptation in one population can even undermine the progress of the other. Another negative alternative to an arms race is *collusion*, or the occurrence of a *mediocre stable state* (MSS) [3, 27], where the individuals interact without actually creating improvement in one another with respect to an objective problem solution. The manifestation of the Red Queen effect in the Standard Adaptive Mapping DGP will be discussed in the Chapter 3.

Theoretical analysis in the last decade has typically shown that cooperative coevolution evolutionary algorithm (CCEA) implementations are not well-suited for static optimization problems [93]. The problem is that CCEA are not necessarily attracted to the optimal collaboration, or finding the team composed of individuals with the optimal performance, in static optimization tasks. This pathology can be referred to as *relative overgeneralization* [71]. A response to this for researchers Wiegand and Potter [93] was to point out very good performance for multiagent learning problems

(although Weigand also continuing to improve CCEAs for static optimization problems [71, 72]). CCEAs are able to perform well in multiagent domains because the goal of the problem is not to find the optimal team, but to find a team that performs well and is robust to deviation in individual members. In other words, CCEAs are well suited to finding individuals that combine well with a variety of individuals from the other populations.

Other researchers have currently been working on modifying CCEAs to enable them to perform well on static optimization problems. Bucci and Pollack [18] have attempted to accomplish this using a combination of Pareto selection and memory mechanisms. Experimenting on two populations, a somewhat competitive angle is taken on CCEAs. One population is cast as the candidate solutions (learners) and the other as the tests for the solutions (evaluators), with both populations taking turns alternating roles as candidates and solutions. Using a steady state tournament size of 2, instead of adding the individual with the highest objective fitness function to the next generation, the algorithm adds the Pareto dominant individual to the next generation if a threshold is exceeded, otherwise the lower individual is added. Both individuals are added in the case of incomparable individuals. Furthermore, the Pareto front of each current population is carried forward to the next, with empty spaces filled using the selection mechanism just mentioned. The highest valued individual of each generation is always carried forward to the next population as well. Given this algorithm, the authors were able to find the global optimum of a popular *maximum of two quadratics* (MTQ) optimization problem. MTQ is a class of functions simply defined as $max(f_1, f_2)$ where f_1 and f_2 are two quadratic polynomials.

Panait, Luke, and Wiegand [71, 72] improve CCEA for the same MTQ problem using a different method. In their approach, a member of a population is evaluated in the context of co-evolving members in each of the other populations (as in a typical CCEA), but the choice of collaborators is biased toward a globally optimal collaboration. A hurdle to overcome for this approach was that the authors had provided the optimal assessment for a population to the algorithm *a priori* in [72], where this information would not be available in most real world domains. The authors, however, have very recently improved the algorithm in [71] to include an implicit memory mechanism by embedding the populations in spatial geometries and choosing breeding and collaboration individuals using a notion of neighborhood. A rote learning algorithm to learn biasing information was also implemented to create a more realistic implementation: if an individual picks an action *a*, the collaborator picks an action *b* that has so far exhibited the highest performance when paired with *a*. The work in [71, 72] is also based on the assumption that a particular type of problem called a multipopulation symmetric cooperative coevolutionary algorithm (MPS-CEA) is being investigated, where for any collaboration used to evaluate an individual in a particular population, if the same collaboration were used to evaluate any of the other component individuals in the set of collaborators, those individuals would receive an equal payoff.

2.1.5 Alternative Mappings in Coevolution

In sections 2.1.3 and 2.1.4, we looked at alternative GPM structures, types of coevolutionary algorithms, and pathologies of coevolution. This section discusses other coevolutionary approaches that involve evolving GPMs in a population, bridging the fields of GPMs and coevolution covered in the previous two sections. The structures of the mappings discussed in this section, however, differ from genetic code mappings that directly relate genotype codons (sequences of genotype bits) to function set symbols as in this work and the more related works of Keller and Banzhaf and Margetts and Jones. The work of those two groups thus warrants separate and more detailed discussion, which follows in Sections 2.2 and 2.3, respectively.

An alternative (not genetic code-based) GPM structure is used in the coevolutionary system SYMBIOT by Paredis, which was found to be capable of better fitness performance on a 3 bit deceptive problem [76] and problems of increasing degrees of epistasis [74] than single population approaches. The 3 bit deceptive problem was described by Goldberg in [31], where deceptive problems are defined as those that tend to lead an EA toward a particular local optimum, when in fact the global optimum is located at that local optimum's complement. Epistatic problems involve closely coupled alternatives during solution search, and typically result in an inability to decompose a problem into independent subcomponents. Epistatic problems are thus difficult for GAs because most GA algorithms rely on forming solution subparts (or "building blocks") and combining their features to form solutions. Paredis's SYMBIOT system overcomes these difficulties by using a population of genotypes and a separate population of permutations

of the genes. It is thus a cooperative coevolutionary implementation, since members of two populations must interact to generate a phenotype. A mapping involves using the permutation to sort the genes into an ordered genotype, and the ordered genotype in turn is evaluated with the objective fitness function. Thus, the mapping that is evolved by coevolution is actually only a specification of the ordering of genes, rather than the definition of the correspondence between genes and members of a function/terminal set.

Every tournament round involves two parent genotypes being selected (where selection is always biased toward higher fitness) and their application to a permutation. The parents then generate a child, which is also applied to the permutation to give it an initial fitness value. For each round, the permutation is given a fitness that is the mean fitness of the parents divided by the child's fitness. Each permutation is given a lifetime fitness based on the mean of the last 20 collaborations in which it was involved, a technique known as Life-time Fitness Evaluation (LTFE). Reproduction of permutations occurs every 20 collaborations, where a special genetic operator used to inherit gene adjacency information is used to form the child. An initial fitness is determined by having the permutation child interact with two solutions, and the 20 LTFE elements of the child are initially given that fitness value.

The LTFE is a measure of the partial fitness history of an individual, and is continuous in that it is updated with new collaborations. LTFE is provided as an effective means of overcoming noise in fitness evalution during steady state tournaments due to the continuous feedback it provides to average out fitness evaluations [74, 76]. LTFE is also claimed to be well suited to coevolutionary coupled fitness landscapes, where changes (through ranking or new individuals) in one population affect the fitness

of new members of the other population [74, 76]. However, later work (also by Paredis) shows that LTFE in a coevolutionary algorithm is unable to overcome the Red Queen Effect [75]. To combat the Red Queen Effect, the algorithm in [75] is changed by not allowing one population to evolve. Instead, a member is selected randomly (rather than according to fitness) for removal from the population every round and a new, randomly generated, member takes its place. By stopping the evolution of one of the populations, the two populations were no longer able to cause adverse effects on one another in their pursuit of higher fitnesses. However, the reader should also note that obviously the opportunity for the evolution of one population to push the evolution of the other forward is also lost.

A coevolutionary model such as Paredis's SYMBIOT enforces an explicitly hierarchical relation between the two populations, with fitness evaluation being performed on one population and a mechanism defined for propagating fitness back to the second population. Such a scheme is generally referred to as symbiotic cooperative coevolution [64]. The symbiotic model of coevolution has been demonstrated to be particularly effective when evolving neural networks [63]. Specifically, one population represents neurons whereas the second defines the connectivity of neurons defined in the first. Fitness is evaluated at the second population and propagated back to the first to establish fitness at the neural level. No recognition of the potential negative contribution of coevolutionary factors such as the Red Queen effect was made. Interestingly, a later work [89] recognizes that solving the gene context problem is central to providing an effective model for coevolution. However, the solution adopted relies on the use of

historical markings in order to enforce suitable speciation policies, where search operators operate within the same species.

The symbiotic model will also be used in this work; however, we explicitly address the Red Queen effect, which may detract from the desirable properties of the cooperative coevolutionary paradigm, as demonstrated in Chapter 3. It should be noted that Paredis appears to be only author (aside from the current thesis) that has attempted to tackle coevolutionary design pathologies while also investigating the production of effective genotype-phenotype mappings (although her mappings simply specify orderings rather than modeling the genetic code). Later works by Paredis [73, 77] moved away from studies with SYMBIOT and cooperative coevolution to investigate overcoming loss of gradient in competitive coevolutionary problems through maintenance of partial or entire fitness history of individuals and a mechanism called "balanced coevolution" that gradually restricts the reproductive rate of one population as it outperforms the other.

Similar to Paredis's SYMBIOT, Murao *et al.*[65] use cooperative coevolution of one population of genes and another population dictating permutations of the genes. As in the mapping mechanism of SYMBIOT, the genes are ordered by a permutation, and the newly ordered genome is then applied to a fitness function. Murao *et al.*'s system did differ from SYMBIOT in that LTFE was not used. Instead, the fitness of a genotype is determined by applying it to all permutations, and the maximum of those values is taken to be the fitness of the genotype. This greedy method of fitness evaluation only allows individuals to evolve in the context of the best individual in the other population, which (in the author's opinion) is a poor choice of fitness evaluation if that best individual is expected to change at all. At that point, the context against which the original population

has been evolving will be lost and fitness will plummet in the original population. This situation, a manifestation of the Red Queen Effect, will be discussed in much greater detail in Chapter 3 in relation to Margetts and Jones work. The fitnesses of the permutations are determined using the same methodology. Also, SYMBIOT used a steady state tournmament whereas Murao *et al.*'s algorithm generated every combination of genotype and permutation and ranked them all in each round of a generational tournament. The problem domain was again a 3 bit deceptive problem, where Murao *et al.*'s coevolutionary system was shown to outperform a traditional GA.

Bui *et al.* [19] have cooperatively coevolved two populations of matrices for use on rotated problems, or problems where a function ought to undergo a transformation operation with a fixed rotated matrix in parameter space prior to being evaluated using the fitness function. One population is the genotypes and the other population is considered to be the mappings. To evaluate the combination of genotype and mapping, the two matrices are multiplied and the result applied to the fitness function. Evaluation of an individual in either population involves pairing it with the best individual in the other population, which (as mentioned above) is not robust to the issue of loss of context of the best individual in the alternate population.

O'Neill and Ryan have created an implementation of Grammatical Evolution (GE) that builds on the system described in [68] by cooperatively coevolving a population of genotypes and a population of mappings. According to O'Neill and Ryan in [69], the only study previous to their work at the time (2004) on evolution of genetic code in a developmental GP is that of Keller and Banzhaf [45, 46]. The author agrees that this is true if the term *genetic code* must model the natural genetic code in that it is

redundant, otherwise the efforts of Margetts and Jones [57-59] using a one-to-one Huffman-based mapping in a developmental algorithm ought to be included as background. The algorithm is called "grammatical evolution by grammatical evolution" or $(GE)^2$. Their algorithm coevolves two populations, one population of solution grammars and another population of solutions themselves. By modifying the grammar used to construct a solution, the algorithm is directly modifying the genetic code by altering the mapping of codon values to different rules in the grammar. A generational tournament is used, with each solution individual being paired with its own solution grammar. Crossover only occurs between individuals of the same type; that is, solution grammar chromosome segments are not exchanged with solution segments. To be precise, $(GE)^2$ actually uses two distinct grammars, a static *universal grammar* and the *solution grammar* that evolves. A globally applicable set of syntactic rules in the universal grammar dictates the construction of the evolving solution grammars. A portion of every solution grammar is hard coded such that evolution occurs on the number of productions for specific nonterminals, with rule duplication permitted. Thus, $(GE)^2$ has the potential to produce redundant (degenerate) genetic codes. The initial work on $(GE)^2$ [69] demonstrated that coevolution of genetic code (grammar) and solution was possible on both static and dynamic symbolic regression problems. Follow-up work in [67] incorporated the principles of modularity and reuse into the universal grammar by constructing the universal grammar so that it dictated building blocks of various sizes be produced. In other words, the universal grammar specifies the construction of another generative bitstring grammar that is used to incorporate building blocks in the generation of the phenotype. The grammar-based mappings of O'Neill and Ryan are versatile and

novel, but this work is based on more direct "genetic code" type mapping constructs such as those of Keller and Banzhaf [5, 45-47] and Margetts and Jones [57-59]. Their work is now discussed in greater detail (Sections 2.2 and 2.3).

2.2 Developmental Genetic Programming with Evolved Genetic Code Mappings

2.2.1 Evolved Genetic Code Mapping Structure

Banzhaf and Keller's seminal work in [5, 45, 47] expands on DGP variants previous to their work which used a "global code" where all genotypes are mapped onto phenotypes using the same mapping. Banzhaf and Keller separate the genotype and phenotype spaces with a genetic code-based GPM (genotype-to-phenotype mapping) that directly maps bit sequences present in the genotype to members of the phenotype's function and terminal sets. Their DGP algorithm evolves both genetic codes and genotypes, coupled together as a single individual, with the aim that artificial evolution will produce genetic codes that support the evolution of good genotypes.

In their model, a phenotype is a syntactically legal symbol sequence where every symbol is a member of the function set F or terminal set T. For a regression problem example, for instance, a member of the phenotype might be a legal sequence where {*} comes from F and $\{x, y\}$ comes from T. The genotype is composed of $n > 0$ contiguous bit sequences each with the same length $b > 0$ called codons. The length b is selected such that for each symbol of the function and terminal set, there is at least one codon to map to that and only that symbol. For instance, a genotype with $n = 5$ and $b = 3$ could be 101 011 110 001 010 and encode the symbol sequence $\{x * y + /\}$. The genetic code is

the codon-symbol mapping that defines the encoding of each symbol by one or more codons. The only remaining piece to create a genotype-phenotype mapping (GPM) is to make sure the raw sequence of symbols that the genetic code encodes is actually a legal sequence. (Only legal sequences are members of the phenotype.) An illegal syntax error in a position of a symbol sequence is fixed using a repair algorithm such as replacing repair [47] or deleting repair [45]. (Repair mechanisms are not required in the further work of Margetts and Jones [57-59] or the algorithm presented in this thesis, and thus they are not detailed in this survey.) Following repair, the genotype is then mapped to a member of the phenotype, and the GPM is complete.

In terms of solution search, the purpose of evolving the genetic code is a means of enhancing the search process so that it profitably adapts the fitness landscape. Evolving the genetic code allows the implementation to isolate phenotype symbols (i.e., instructions) that are valuable in solving a particular problem and use them more often, while using irrelevant symbols less often. In addition to the traditional genotype population, a population of individual genetic codes is used instead of a single global code. Each individual is a coupling of its own genetic code along with its genotype. The individual's code is an arbitrary codon-symbol mapping, and thus may map more than one codon onto the same symbol. Redundant code can thus be used to produce a phenotype in their implementation, where the code redundancy allows the search to emphasize certain symbols and disregard symbols that are of little or no use in the problem solution. The structure of a global and non-global code and the encoding process involved in each are given in Figure 2.2.

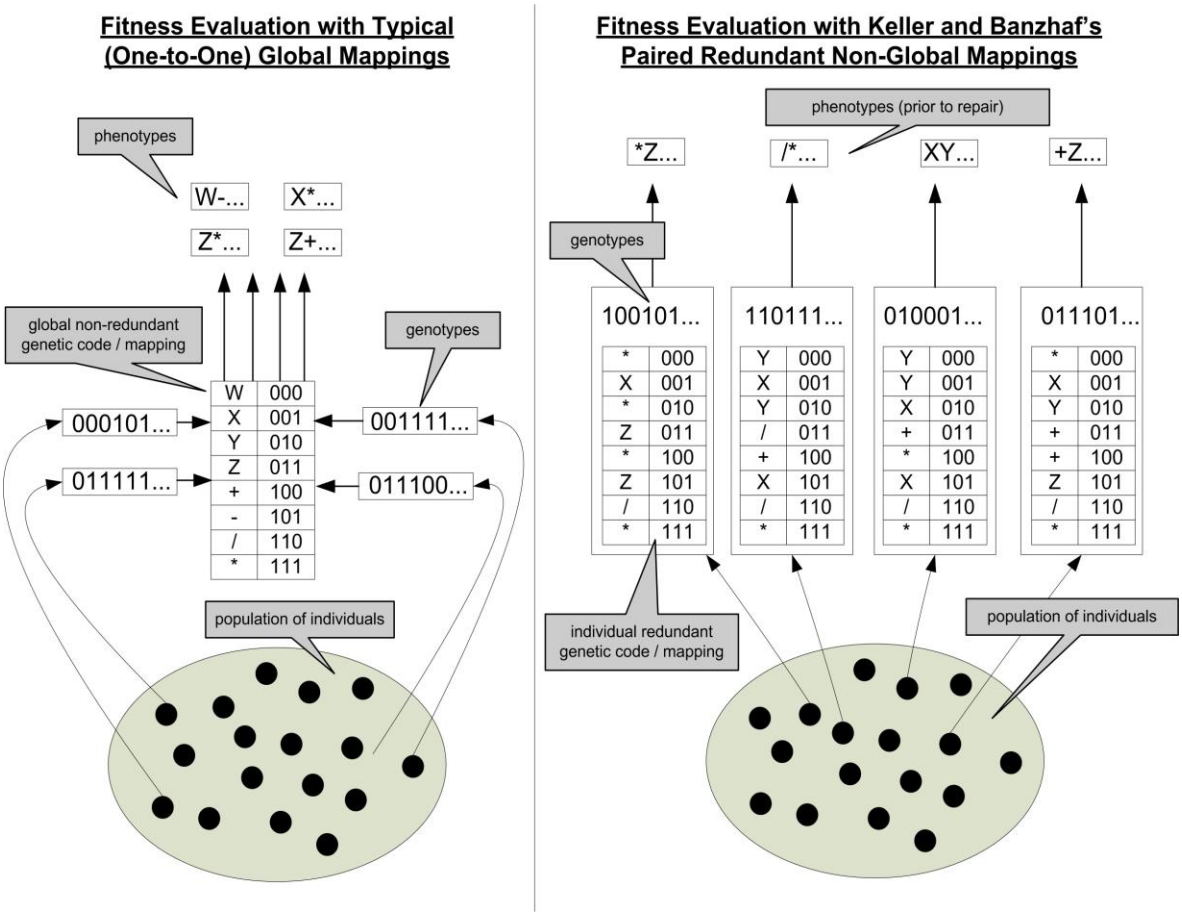


Figure 2.2. Fitness evaluation using typical global and Keller and Banzhaf's non-global genetic code-based mappings.

2.2.2 Evolved Genetic Code Mapping Algorithm

Having established the nature of the mapping individual's structure and encoding, the components necessary for evolution of the code population in the implementation of Keller and Banzhaf [45] are defined in this section. Mutation for the code population in is point code mutation. For Keller and Banzhaf, a symbol in the mapping is stochastically selected and then replaced with a different symbol that is stochastically selected from the symbol set. The mutation probability for the point code can be set at a different rate than the mutation rate for the individual's genotype. A code population individual is reproduced when its connected genotype individual is reproduced, and different genotype individuals are permitted to carry the same code. The authors use a generational tournament with fitness-based selection and a tournament size of two. Individuals are stochastically selected, and point mutation or simple reproduction is performed based on an execution probability (no crossover is used).

The genetic code carried by an individual determines, along with the genotype, the fitness of the individual's phenotype. However, since the same genetic code carried by an individual with a different genotype will result in a potentially very different fitness for the individual's phenotype, the authors introduced an additional fitness measure for genetic code [45]. The code fitness measure relied on enumeration of the search space and *a priori* knowledge of an ideal solution for a simple regression problem. The fitness of the genetic code is then defined as the portion of the search space that it maps onto an ideal solution under the GPM, including repair algorithm. So if a genetic code maps 10 genotypes out of a search space of 50 to the ideal solution, the fitness is 0.2.

Using both phenotype fitness measure and code fitness measure simultaneously, [45] demonstrates that evolution works in principle on an easy synthetic problem. Research in [45] then extends the fitness measure to respond to the question of whether better individuals have better codes and vice versa. A notion of coupled fitness is defined simply as the product of an individual's genotype fitness and their code fitness. Using this measure, the authors find that better individuals have better codes when analyzing the behavior of coupled fitness using selection based on the previous fitness definition. The authors attribute this code evolution to the propagation of codes defining better individuals, who propagate their attached codes. The algorithm is extended to a harder synthetic problem in [46]. The fitness determination and selection procedure for their algorithm is shown below in Figure 2.3, with the associated pseudocode for their procedure in Figure 2.4 [45].

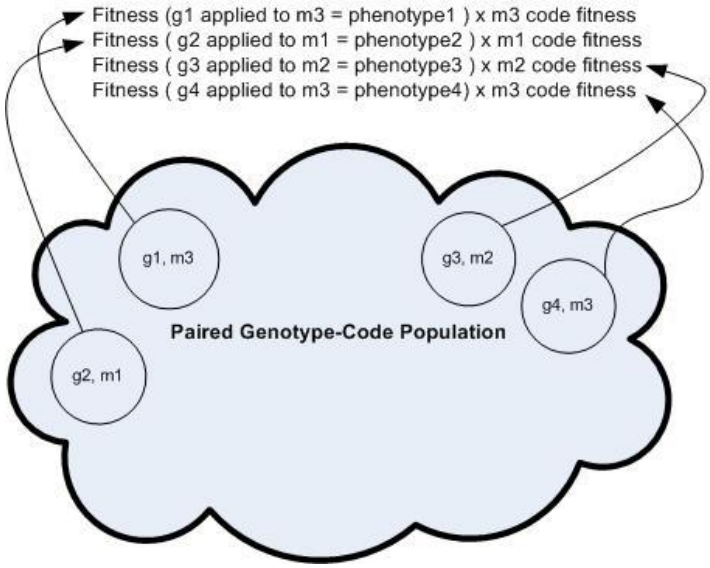


Figure 2.3. Selection mechanism for Developmental GP using Evolved Mappings.

```
Stochastically generate initial population of genotypes.
Apply GPMs to genotypes to create phenotypes.
Apply fitness function to population of phenotypes.

while (desired fitness not found)
   Select 2 genotypes based on fitness. (i.e. this is tourn. size)
   Reproduce or apply point mutation to selected genotypes.
   Apply GPMs to genotypes to create phenotypes.
   Evaluate fitnesses of selected individuals.
   Insert back into population.
```

Figure 2.4. Algorithm for Developmental GP using Evolved Mappings.

2.3 Developmental Genetic Programming with Adaptive Mappings

2.3.1 Adaptive Mapping Structure

Keller and Banzhaf's DGP [45] is composed of a single population of genotypes and an attached genetic code, Section 2.2. In contrast, the DGP of Margetts and Jones described in this Section separates the mapping and genotypes into two populations in order to implement cooperative coevolution. In [57], Margetts dubs the approach of each individual having their own mapping function the *monolithic* approach. According to Margetts, one downside of this approach—as opposed to placing genotypes and mappings in separate populations—is that a single population is composed of individuals containing significant amounts of genetic material. Also, each individual must effectively solve two problems at once: it must simultaneously determine a good mapping and it must find a good solution to which the mapping is to be applied. Supposing that an individual is composed of a good solution and a poor mapping (or vice versa), the individual will only possess a single fitness value. There would be no way of

knowing that a superior solution and poor mapping (or vice versa) are appended to one another without separating them anyway.

Margetts goes on to say that the only way that an individual can achieve a high fitness is if both mapping and solution components are of high fitness. He suggests in [57] that it may seem (at least on the surface) that this problem could readily be alleviated by specialized genetic operators. But supposing that an operator guaranteed that a child inherits either the mapping or solution component of the parent and the remainder of the child genotype was generated with a crossover type operator, although the child has inherited a reasonable solution or mapping, it will likely still not have both at the same time. After all, the ability to alter the mapping on an individual basis means that different individuals may represent entirely equivalent, yet entirely incompatible, high fitness solutions. By modeling genotype and mapping as separate individuals, genotype solutions can evolve in the context of particular mappings.

In addition to modeling the placement of the genetic code as a separate entity instead of inclusion in the genotype, the structure of the genetic code itself is also addressed by Margetts and Jones in [58, 59]. Earlier work allowed bit strings of equal lengths as codons and used the redundant encoding of symbols in order to emphasize and de-emphasize certain symbols. Margetts and Jones chose an alternative representation in an attempt to improve the symbol emphasis component of DGP search. In [58] they introduce their alternative representation where they use binary strings of dynamically determined length each corresponding to a symbol, although all symbols need not be encoded. The algorithm chooses its own one-to-one assignment of binary sequences to symbols using Huffman encoding. (As we noted previously, such a non-redundant

encoding is unfortunately not developmentally accurate.) Huffman encoding is traditionally a means of allowing frequently-used symbols in a message to be transmitted using a proportionally lower number of bits so that message lengths are reduced for the purpose of compressed transmission. The Huffman algorithm also ensures that for a given bit length, the most frequent bit sequences correspond to the most frequent symbols. The pseudocode for the Huffman algorithm as presented in [82] is given below in Figure 2.5.

```
for (i = 0; i <= noInFunctionSet; i++)
     if (count[i] >= 1) priorityQueue.add(count[i], i);

while (!priorityQueue.isEmpty())
     i++;
     t1 = priorityQueue.remove();
     t2 = priorityQueue.remove();
     parent[i] = 0;
     parent[t1] = i;
     parent[t2] = -i;
     count[i] = count[t1] + count[t2] ;
     if (!priorityQueue.isEmpty())
          priorityQueue.add(count[i], i);

for (j = 0; j <= noInFunctionSet, j++)
     j = 0; x = 0; k = 1;
     if (count[j])
          for (t = parent[j]; t; t = dad[t] ; k += k, i++)
               if (t < 0)
                    x += k ;
                    t = -t ;
     code[j] = x ;
     len[j] = i ;
```

Figure 2.5. The Huffman encoding algorithm.

In the pseudocode above, the algorithm starts with an array "`count`" that is filled with the frequency counts for each of the members of the functional set. (Typically

Huffman's algorithm would have to count the number of occurrences of each character token in a message and produce the count array itself. In the adaptive mapping algorithm, the mapping individual itself provides the frequency information.) The first statement in the pseudocode above inserts all nonzero frequency counts into `priorityQueue`. Next, inside the while loop, the two smallest elements are removed from the priority queue, their counts are added together, the result is placed back into the priority queue. The condition of the while loop produces a tree of frequencies, where `parent[t]` holds the index of the parent of the node whose frequency is in `count[t]`. The sign of `parent[t]` indicates whether the node is a left or right child of the parent. The final for loop in the pseudocode creates the encoded representation of the function set, represented by the arrays `len[]` and `code[]`: the rightmost `len[j]` bits in the binary representation of the integer `code[j]` is the code for the `j`th function in the set. For instance, the function ADD could be the fourth function, so `j = 4` and `code[4]` = 6 with `len[4]` = 3, indicating that the code for ADD is the rightmost 4 bits of the binary representation of the number 6, or `0110`. (Note that 0s are prefixed if j exceeds the length of the binary representation of `code[j]`.)

The Margetts and Jones algorithm that uses the Huffman representations is described in [57, 58] and introduces the concept of an "adaptive mapping." Given a function set with *s* symbols, each individual in the population of mappings consists of *s* 10-bit binary string segments each representing a frequency. Each segment of 10 bits is converted to a real valued frequency in the interval [0, 1] using the normalized *countingOnes* function that simply sums all the ones in a given binary string:

$$countingOnes(binaryString) = \sum_{i=0}^{L-1} binaryString[i]. \quad (2.1)$$

where *binaryString* is an array of binary characters and *L* is the length of *binaryString*. In [59], the use of the *countingOnes* function is justified over more intuitive choices (such as binary to decimal conversion) for its ability to produce small changes in phenotype to correspond to small changes in genotype in later phases of search. The symbols, associated frequencies, and genotype are provided as arguments for utilizing Huffman's algorithm, and it returns a one-to-one mapping of varying-length bit strings to symbols. The Huffman mapping encoding process is depicted in Figure 2.6. In [58], the authors demonstrate that their algorithm and associated adaptive Huffman-encoded mapping outperform a fixed mapping on a DGP variant of the Maximum Output problem. As mentioned in Chapter 1, the adaptive mapping and associated algorithm will be denoted the Standard Adaptive Mapping (or simply Adaptive Mapping) DGP in this thesis.

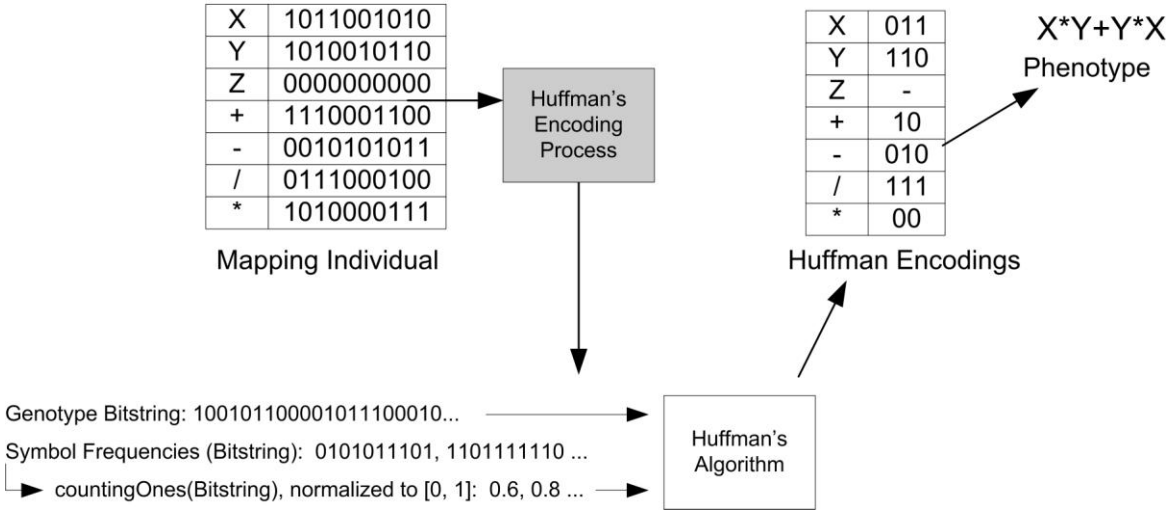


Figure 2.6. The Adaptive Mapping structure and encoding.

No repair mechanism is introduced for the Adaptive Mapping structure [57, 58]. The way in which the genotype is parsed and the nature of the mapping does not create illegal sequences. The number and sequence of bits to be read from the genotype to represent a member of the functional set is always changing under Huffman's algorithm, but the encoding always corresponds to a function. The appropriate number of bits is first parsed corresponding to the function, and then any required bits needed to address a target or source register, retrieve a variable from the problem definition, or generate a constant, are parsed from the genotype. A genotype string is continually parsed until there are insufficient bits to parse the next instruction. If there are insufficient arguments for a function, it does nothing. Given that a program is a series of function calls, guaranteed to execute only if there are appropriate arguments, there are no illegal symbol sequences generated. Thus, the Adaptive Mapping method does not require a repair phase.

We can explore why this is the case in a bit more detail, since Yu and Bentley [103] have identified a classification system for methods used to ensure that phenotypes are legal (satisfy problem constraints). Eleven methods/classifications are discussed, each corresponding to the stage at which they are incorporated into the design and execution of an evolutionary algorithm. Using their classification system, the adaptive mapping using Huffman encoding does not restrict the search space, since genotypes are always interpreted in a different way and cannot be designed to be only capable of representing legal solutions. It therefore does not fall under the "legal search space" classification, where constraints are addressed at the very beginning of the design of the algorithm. This is beneficial, since it allows free exploration of the genotype space

without avoiding infeasible regions, which Keller and Banzhaf recognize as one of the main benefits of separating genotype and phenotype spaces [47].

Instead, the function set for the phenotype itself does not allow illegal phenotypes. Phenotypes using these adaptive mappings are sequences of functions with accompanying arguments, and no sequence of functions is invalid. This mapping thus uses Yu and Bentley's "legal solution space" method of satisfying constraints, where a legal phenotype is provided during the actual design of the phenotype. In addition, the adaptive mapping using Huffman encoding of course provides the mapping rules that transcribe the binary content of the genotype to a phenotype (which we have already stated is legal). The adaptive mapping thus uses the "legal map" classification of constraint satisfaction: The phenotype solutions produced from the mappings applied to genotypes satisfy constraints, but genotypes are not disrupted or constrained because they are not directly connected to the phenotypes due to the intermediate mapping. In summary, the adaptive mapping uses a legal map to a legal solution space to separate phenotype and genotype spaces without need for repair, allowing unrestricted evolutionary search of the genotype space.

2.3.2 Adaptive Mapping Algorithm

The Adaptive Mapping DGP algorithm uses a population of mapping individuals and a population of genotype individuals. The two populations co-evolve so that a search is conducted for both a useful mapping and a desired solution simultaneously. In order to evaluate a particular genotype, a member of the mapping population is selected to produce the phenotype for fitness evaluation. The mapping individual chosen for

application to a genotype is the mapping individual with the current highest fitness in the population. To evaluate a mapping individual, the current best member of the genotype population is used. There is thus only one mechanism for fitness evaluation for genotype and mapping in this implementation: fitness of both mappings and genotypes is evaluated given a phenotype produced from a mapping individual applied to a genotype individual. The authors speculate that the best way to evaluate the fitness of a mapping individual would be to evaluate it in the context of all genotypes and take the average fitness as the fitness of the mapping, but acknowledge that such measures are too computationally expensive to be feasible.

In [57], the algorithm is explained in greater detail than in [58]. The fitness of a member of one population cannot be evaluated without a member of the other, and there is initially no information about the best mapping or genotype to use. The members of each population are thus evaluated with random members of the other to accomplish initialization. A steady state tournament is then used with members of each population evaluated alternately. In the Adaptive Mapping algorithm, two parents are chosen from a population and the mutation and crossover operators are applied to them to create children. The children are evaluated using the best member of the other population. The children then stochastically replace individuals in the population selected with a probability inversely proportional to fitness. For the purposes of implementing the Standard Adaptive Mapping DGP in this thesis, we use a steady state tournament that selects four individuals each round, replacing the worst two individuals with children of the winning two parents (parent copies subject to mutation and crossover based on associated thresholds). This approach is comparable, but more straightforward. Also, it

protects the best individual in either the mapping or genotype population at any given time, whereas the approach of replacing individuals in a population selected with a probability inversely proportional to fitness does not. As we will see in the upcoming chapter, the Standard Adaptive Mapping algorithm requires every benefit it can get in terms of protecting its best individual to the greatest degree possible. In other words, this subtly different (but still steady state) selection mechanism is intended to cast the Standard Adaptive Mapping algorithm in the best possible light. The selection mechanism is depicted in Figure 2.7 below, with pseudocode provided in Figure 2.8.

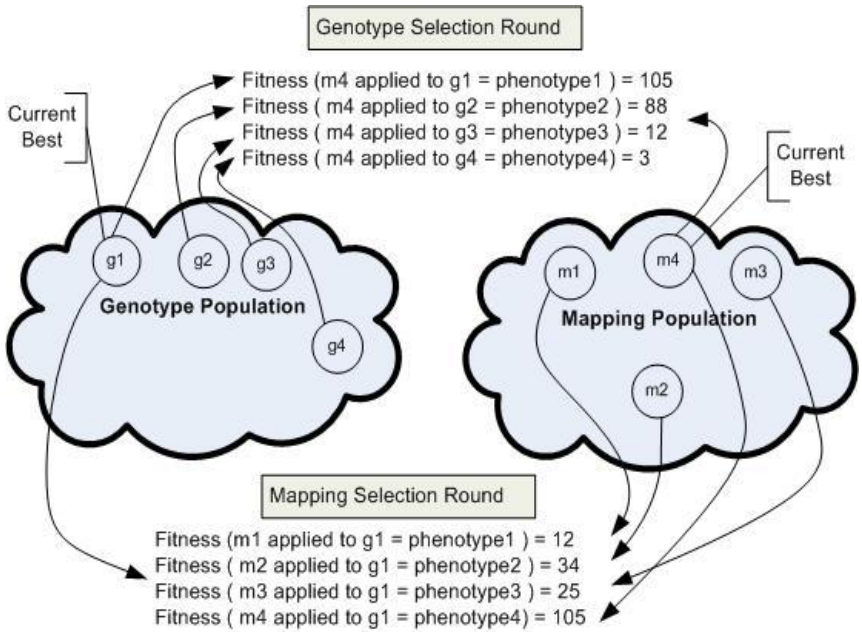


Figure 2.7. The Standard Adaptive Mapping algorithm selection mechanism.

```
Stochastically generate initial population of genotypes.
while (tournamentNotDone && solutionNotFound)
   Select 4 genotypes with uniform probability
   Rank selected genotypes using bestMapping
   Verify or set current bestGenotype
   for 2 loserGenotypes
      if (mutation threshold is met)
         mutate(loserGenotype = copy of parent)
      if (xover threshold is met)
         xover(loserGenotypes = copy of parents)
   Select 4 mappings with uniform probability
   Rank selected mappings using bestGenotype
   Verify or set current bestMapping
   for 2 loserMappings
      if (mutation threshold is met)
         mutate(loserMapping = copy of parent)
      if (xover threshold is met)
         xover(loserMappings = copy of parents)
```

Figure 2.8. The Standard Adaptive Mapping Algorithm.

2.4 Where this Work Stands on the DGP Design Issues

It should be noted that this work will show that neither the Adaptive Mapping Structure described in Section 2.3.1 nor the algorithm that accompanies it (Section 2.3.2) are a good choice for many regression and classification problems. The design issues surrounding the mapping will not be discussed until Chapter 6, but we believe that redundancy in mappings (as implemented by Keller and Banzhaf) is a better alternative than Huffman encoding for most interesting problems (for performance reasons, as well as theoretical and developmental arguments). However, addressing this problem is not the initial step of this work.

For now, we implement genotypes and mappings as described by Margetts and Jones in [57-59], for we aim to show in the following chapter that their particular coevolutionary algorithm suffers from various inherent drawbacks (some of which are

documented in the coevolution literature). To make sure that the design issues explored are a product of the algorithm process itself and not the structure of individuals, the representation of individuals must be kept constant for now. We also describe how the problematic design issues for their algorithm are solved in our implementation. To be clear, our algorithm adopts a two population coevolutionary approach rather than a pairing of genotype and mapping in a single individual for the benefits as cited by Margetts [57] (Section 2.3.1) and for the developmental coevolution reasons cited in this chapter (Section 2.1.1). While we believe the two population coevolutionary approach to hold more promise than the pairing, we believe that it must be properly implemented to overcome problems of separating the populations. The cooperative coevolutionary algorithm that is proposed in this thesis to create better solutions and mappings represents a novel and efficient new algorithm to overcome the problems that malign the current implementation (and likely other similar approaches).

Chapter 3. Introducing the PAM DGP Algorithm

3.1 Introducing the MAX Problem

The problem selected to introduce the Standard Adaptive Mapping DGP in the literature was a version of the MAX problem introduced in [29, 53], as described by Margetts and Jones [57, 58]. This version of the MAX problem is to create a program that returns the largest value possible using the functional set within the given program size limit (rather than a tree depth limit as in [29, 53]). An ideal solution to the problem is a program that repeatedly duplicates a large number and multiplies it by itself, effectively squaring the number repeatedly as many times as possible. Margetts and Jones actually experiment with five versions of their MAX problem, where the population of genotypes consists of 50, 100, 150, 200 and 250 bit length individuals. By having more bits available for a solution, it is easier to generate a larger number by the end of the interpretation of the individual's instructions. MAX problem difficulty thus decreases with increasing bit length of the genotypes.

In the Adaptive Mapping DGP, the MAX problem is posed to a linear (bit string) stack-based version of GP. Each individual is a stack-based machine composed of a general-purpose stack and an output register [57, 58]. A program that changes the state of the machine is a list of instructions from the function set in Table 3.1 below. The function set can also have default codons that are used for a non-mapping benchmark implementation to which the Adaptive Mapping DGP algorithm performance is compared. (Fixed mappings with the default encodings correspond to evolution without a population of mappings, and thus correspond to a traditional GP.) There is no terminal

symbol set for this problem specification. Each instruction in the program is processed sequentially, with each having an associated function. For each function call, the required number of arguments is taken from the stack, they are presented to the function, and the return value (if any) is pushed back onto the stack. If there are insufficient arguments on the stack, the function does nothing.

Table 3.1. Stack-based GP Maximum Output Problem function set.

Symbol	Function Explanation
Plus	Pop 2 items, add, push onto stack.
Times	Pop 2 items, multiply, push onto stack.
Const	Interpret next 10 bits as number, push onto stack.
Dup	Duplicate item on top of stack, push onto stack.
Pop	Remove item at top of stack.
S2R	Copy item at top of stack into output register; does not affect stack contents.
R2S	Push item in output register onto stack.

3.2 Drawbacks for the Standard Adaptive Mapping Algorithm

3.2.1 Drawback 1: A Manifestation of the Red Queen—Repeated Loss of Context and Fitness Spiking

While the Adaptive Mapping DGP algorithm was shown to outperform the fixed mapping algorithm corresponding a traditional non-mapping GP in [57, 58], we show in this section that there are two major drawbacks associated with the scheme. The implementation uses the cooperative coevolution of a population of genotypes and a population of mappings, as described above in Section 2.1.4. An individual from one population cannot be evaluated without a member of the other population. To evaluate the fitness of an individual in either population, the individual is evaluated with respect to

the best individual in the other population. This is done to avoid the impractical computational expense of taking the individual we want to know the fitness of and evaluating them against every member of the other population. The authors also state in [58] that evaluating an individual in a population with the best individual in the other creates an algorithm that is "pleasantly symmetric."

The first drawback is that the methodology of evaluating an individual in either population with respect to the best individual in the other population leads the algorithm to regularly disrupt the progress it makes toward a solution. The reason for this is that, for a given individual in a population, the context of the best individual in the opposite population changes throughout the algorithm. If a brand new best genotype appears, evolution of the mapping population must start its search anew to match it (with the exception of the mapping individual that caused it to happen). That is, every mapping except the one that produces the best fitness with the new best genotype will now likely produce a poor fitness with the new best genotype. The proceeding evolution of the mappings can eventually create a new best mapping in light of the new best genotype, but the previously best genotypes will then fail in the context of this new mapping. The two populations end up struggling to synchronize their search to each other's best individual while at the same time causing the best individual in each other's population to change. The mutual contexts that created a high fitness can quickly become lost with the uncoupling of the relationship of best genotype to best mapping. The same phenomenon can occur with respect to either population. This is a textbook case of the Red Queen Effect: each population struggles against and undermines the progress of the other instead

of building on one another's progress. The steps in which this happens are depicted below in Figure 3.1.

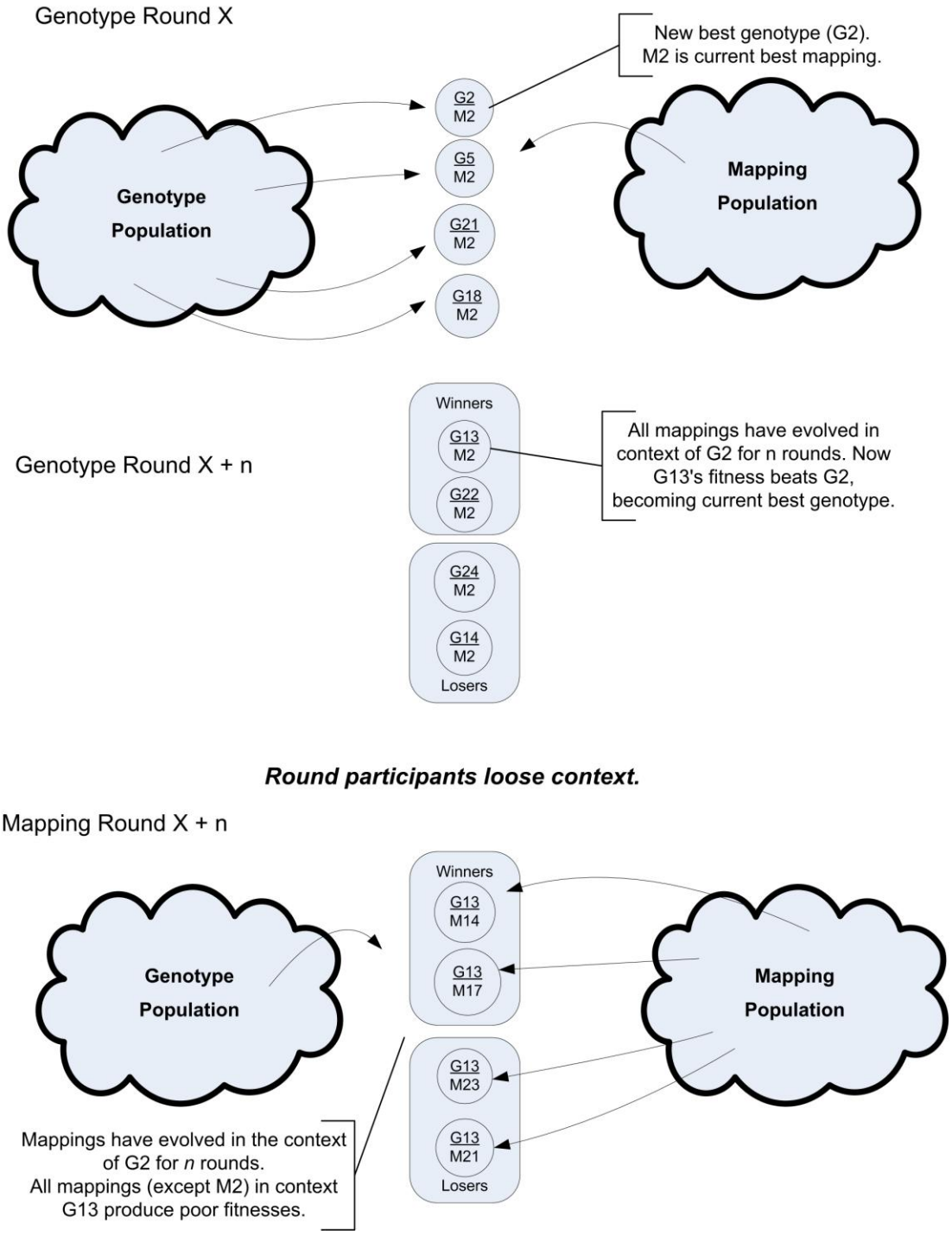


Figure 3.1. Loss of context in the Standard Adaptive Mapping Algorithm.

While elitism to protect the best individuals in ranking ties (or in general) is not a part of the Standard Adaptive Mapping DGP algorithm, it would be an *ad hoc* addition that would likely hinder the exploration phase of the search considerably given the frequency with which context changes were seen to occur. Despite the problems changing contexts cause, the algorithm depends on the changing of the context of the best individual for exploration of the search space to progress toward a solution. Attempting to correct the problem by protecting the best individuals in either population would adversely impact the algorithm. The fitness cycles are a necessary effect of the function of the Adaptive Mapping algorithm, and are seen in a typical run of a tournament below in Figure 3.2 for a population size of 8.

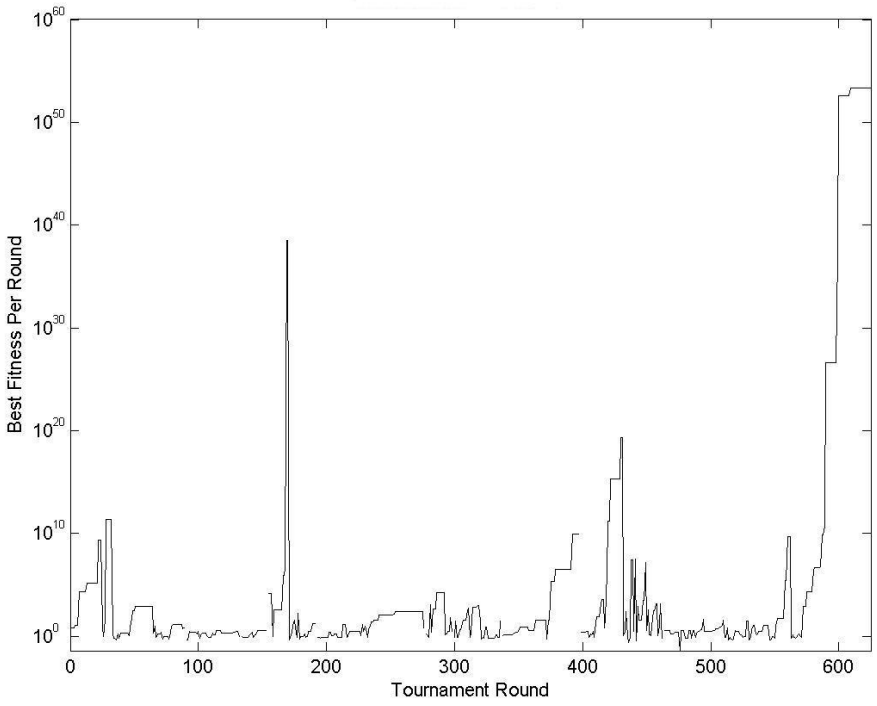


Figure 3.2. Best fitness per round for the Standard Adaptive Mapping DGP for the MAX Problem. This graph represents a typical run for 200 bit individuals and a population of 8. Breaks in the graph indicate fitness points ≤ 1 on the log scale.

Previous results reported for the Adaptive Mapping DGP algorithm do not plot the fitness of individuals participating in each round of the MAX tournament in [57, 58]; only final numerical results are shown for a low number of tournaments. Conversely, data for *maximum* fitness at each round is provided in [57]. However, such a reporting scheme masks the wide variation in fitness resulting from the loss of context between the two populations. As such, steady monotonic improvement in fitness is reported, wherein the fitness spiking of Figure 3.2 cannot be recognized.

3.2.2 Drawback 2: Lack of Exploration of the Search Space

Even if there were not an issue involving constant changes in context, the standard Adaptive Mapping algorithm has a second major drawback that is related to the first. This second major drawback was actually noted before the first was discovered, and was the original motivation for creating a new algorithm to improve on the standard. Since the standard is always evaluating the individuals chosen in genotype tournaments against only one mapping and vice versa, the individuals in either population do not get an adequate chance to evolve in the contexts of other individuals in the opposite population during an arbitrarily chosen stage of evolution. While all members of a mapping population or genotype population may serve their term in establishing a context for the other population, while each is serving their term the members of the other population do not have an opportunity to achieve higher fitnesses by combining with the non-best members. Such combinations could yield higher fitness schemas that go unnoticed in the search conducted by the Standard Adaptive Mapping algorithm, where a greedy mechanism for relating the two populations is assumed to be the best design

choice. The nature of the limited possible combinations of the Standard Adaptive Mapping DGP involved in the search space is depicted below in Figure 3.3.

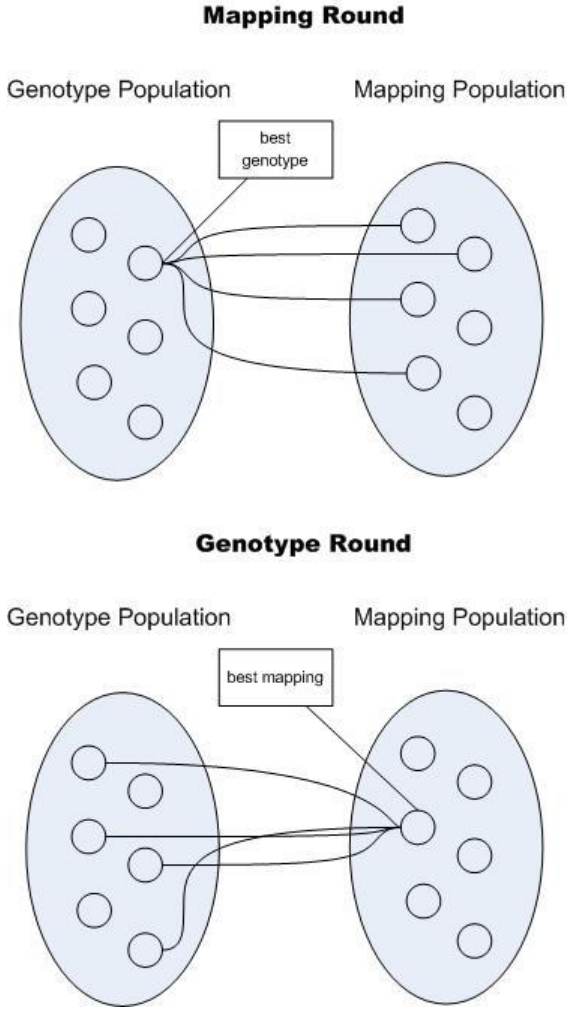


Figure 3.3. Possible combinations during solution search for the Standard Adaptive Mapping Algorithm.

3.2.3 Drawback 3: Lack of Fitness-Based Performance

Experiments using the Adaptive Mapping DGP were reported in [57, 58] to outperform a traditional GP using a default encoding for each function symbol on the MAX problem for most program size limits (bit lengths of individuals). Given the performance obstacles in the Adaptive Mapping algorithm, it is of interest to determine

how it outperformed GP with default encodings (traditional GP with a fixed mapping). The results in [57, 58] were reported for experiments up to 5000 function evaluations, with just 10 experiments run for each of five program size limits (50, 100, 150, 200, and 250 bits). Adaptive Mapping using a limit of 5000 function evaluations leads to a very low number of rounds per tournament: Assuming an average of 3 genotype bits per function call, this meant a tournament length range of 15 rounds (for 250 bit individuals) to 75 rounds (for 50 bit individuals). We decided to run a tournament for more rounds for the Adaptive Mapping algorithm and a fixed, default encoding GP with a population of fifty 250 bit individuals. The results are given in Figure 3.4 below, plotting up to tournament round 20 (where the cited results of [57, 58] go up to 15 rounds for 250 bit individuals). We can see in the graph on the left side that the Adaptive Mapping DGP does indeed outperform the GP with default mappings. However, if the tournament continues just to round 100 (Figure 3.5), it is quite evident that the default encoding GP overtakes the Adaptive Mapping DGP. Loss of context and fitness spiking associated with the Red Queen effect does indeed hinder the performance of the Adaptive Mapping algorithm compared to Traditional GP.

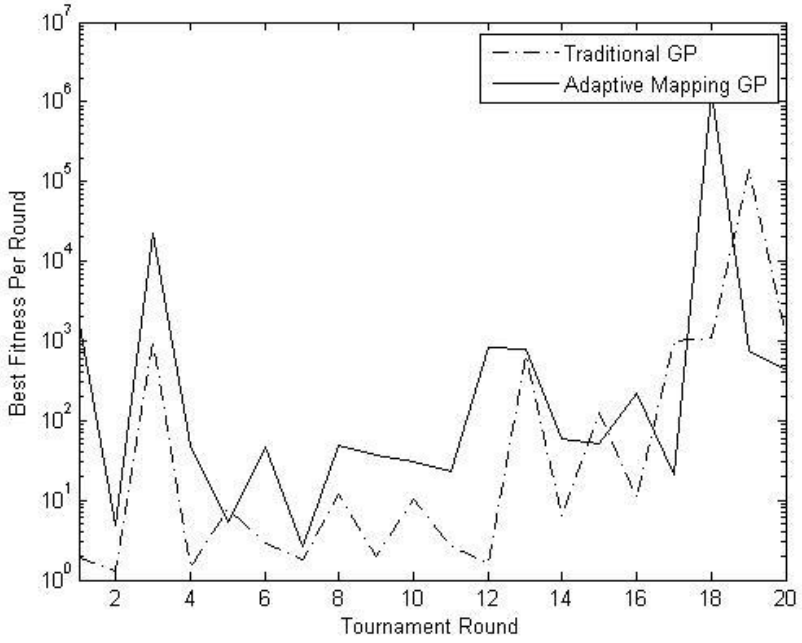


Figure 3.4. Best fitness per round for Traditional GP and Adaptive Mapping DGP for the MAX problem, population of 50 individuals of size 250 bits, up to tournament round 20.

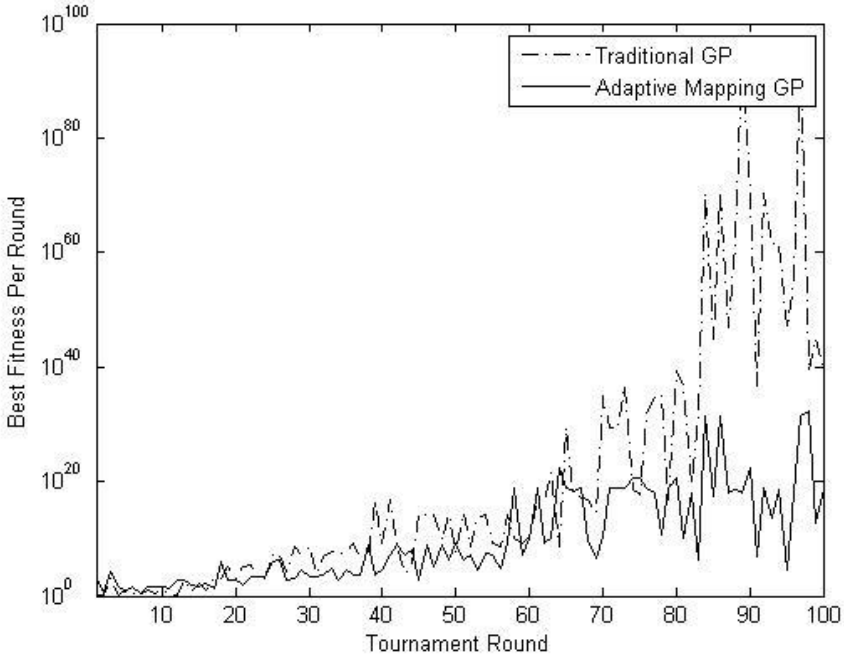


Figure 3.5. Best fitness per round for Traditional GP and Adaptive Mapping DGP for the MAX problem, population of 50 individuals of size 250 bits, up to tournament round 100.

In summary, there is an additional implicit overhead in systems employing evolved mappings in that these systems need to discover both the appropriate genotype and the best function set, whereas traditional GP need only concentrate on locating the relevant genotype. It is thus more difficult problems than the MAX problem, often involving function sets including extraneous symbols, which are best suited to evolved mapping algorithms. Under such conditions, DGP might be able to better the Traditional GP approach by discovering relevant subsets of symbols and concentrating on forming solutions from this subset of the function set, i.e., the size of the DGP search space decreases. The MAX problem will be used hereafter only as a basis to demonstrate how the PAM DGP algorithm overcomes the drawbacks of the Standard Adaptive Mapping algorithm and its associated Huffman-based mapping structure. The purpose of the adaptive mapping algorithm we present in the following section is to create fitted function sets to handle difficult problems, not to solve more trivial problems quickly. Later chapters in this thesis will further improve the algorithm presented in the next section and investigate its application to more challenging problems to which it is better suited.

3.3 The PAM DGP Algorithm

With the above drawbacks of the Standard Adaptive Mapping DGP algorithm in mind, an alternative architecture is proposed in which genotype and mapping population are coevolved through a probabilistic model. Specifically, each individual in the genotype population is allotted a column on the *x* axis of a probability table, and each individual in the adaptive mapping population is allotted a row on the *y* axis. The table is

initialized so that each column sums to unity, or for each genotype each mapping is expressed in terms of a probability. At initialization all cells in each column are assigned the same probability, or 1/(*number of mappings*). This forms a symbiotic model for relating genotype and mapping, where fitness ranking is performed in terms of the genotype but the likelihood of selecting individual mappings is defined by the structure of the probability table. For direct comparison with the Standard Adaptive Mapping algorithm, genotype and mapping structures remain identical. The Huffman based encoding in the mapping population [57-59] is used; however, in Chapter 6, the case of a more developmental adaptive redundant mapping is introduced.

As per the Standard Adaptive Mapping algorithm, a steady state tournament is retained. Individuals are selected using roulette wheel selection to choose a position in the table, with four separate genotype individuals selected in each tournament round, Figure 3.6. Each entry in the table indexes a genotype-mapping pair. The genotypes are ranked by fitness after being interpreted in the context of the corresponding mapping. With a steady state tournament size of four, the two best genotype individuals (the parents) are left untouched while the remaining two (genotype) individuals become children. The children become copies of the parents and are subjected to mutation and crossover with corresponding likelihood probability thresholds. (Crossover and mutation operators and associated rates will follow, being discussed with individual problem parameterizations.) The mappings associated with the ranked genotypes receive the ranking of their partnering genotypes during competition in the aforementioned tournament round (Figure 3.6). The mappings associated with the top two genotype parents are ranked as the top two tournament mappings and kept as the parent mappings,

while the mappings associated with the two (genotype) children become copies of the corresponding parent mappings and are subject to mutation and crossover. Since there is only a guarantee during selection that different genotypes will be selected, naturally, the above process may result in mapping individuals appearing more than once in the ranked list for a tournament round. A mapping appearing twice or more in the ranking list may (and will likely) have a different fitness each time, in virtue of being associated with a different genotype at each placement in the list. If a single mapping is associated with both losing genotypes, the crossover operation is, of course, not performed with respect to mappings for that ranking.

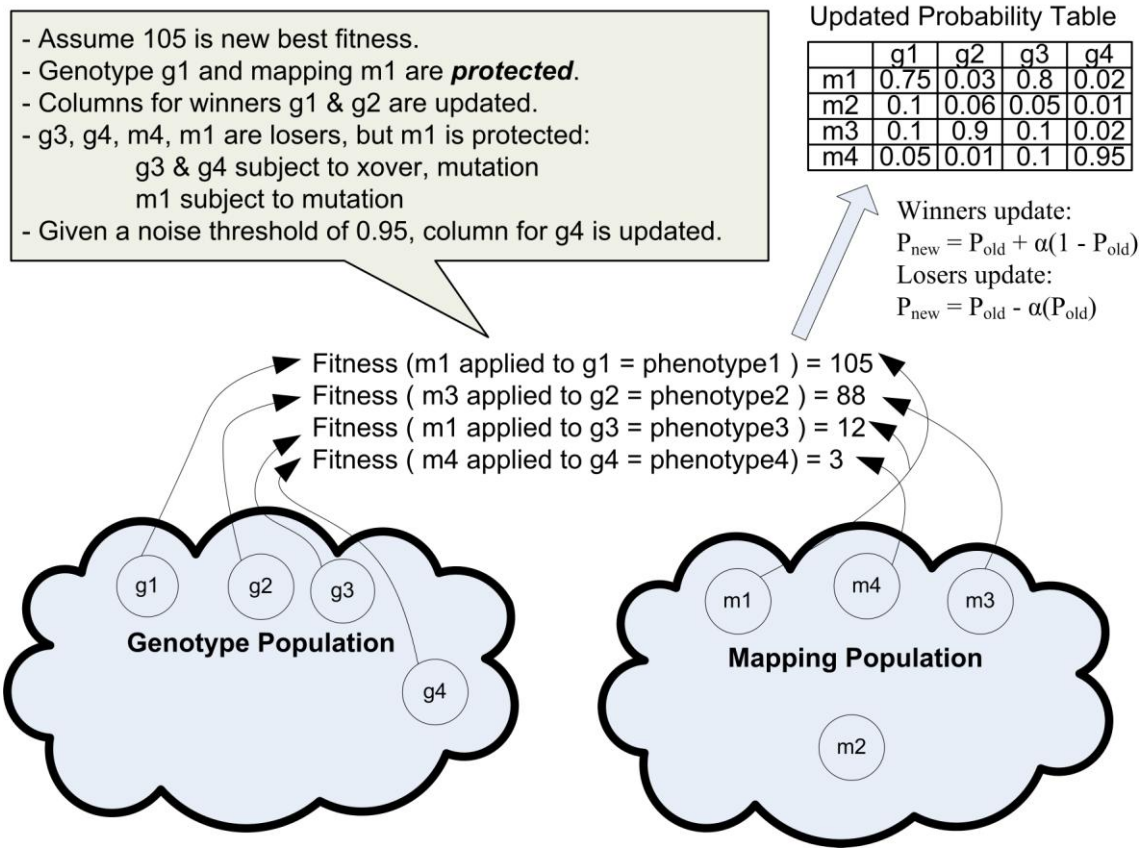


Figure 3.6. PAM DGP algorithm and data structures.

The algorithm also features elitism in that the genotype individual and the mapping that produces the current highest fitness cannot be replaced. Note that unless both members of that pairing are explicitly protected, either member of the pairing can be selected as a child by being coupled with an alternate member of the other population during roulette wheel selection on the probability table. The position on the table associated with the two winning genotype-mapping combinations is updated according to (3.1), while the losing combinations in the same column are updated according to (3.2)[3]

$$\text{Winning Combination: } P(g,m)_{new} = P(g,m)_{old} + \alpha(1 - P(g,m)_{old}) \quad (3.1)$$

$$\text{Other Combinations: } P(g,m)_{new} = P(g,m)_{old} - \alpha(P(g,m)_{old}) \quad (3.2)$$

where g is the genotype index, m is the mapping index, α is the learning rate (or how much emphasis is placed on current values as opposed to previous search), and $P(g,m)$ is the probability in location [g, m] of the table. Recall that four separate genotypes, not necessarily four separate mappings, are chosen per tournament round. Therefore, the genotype-associated columns are updated rather than the rows. Equation 3.1 increases the future probability of choosing the winning combination by a value proportional to the learning rate α and to the previous value allotted to the combination. Given a particular α, smaller probabilities will be increased proportionally more than larger probability values to provide expedient biasing toward selection of newly discovered pairings of greater fitness. Equation 3.2 causes the other values in the column to be allotted a negative reinforcement implicitly by normalization. Roulette wheel selection for the

[3] Equations 3.1 and 3.2 are adapted from [24], where their original use was for an ant-based network routing domain.

probability table operates by adding the probabilities across the rows associated with mapping individuals, although the manner in which the table is traversed does not matter because each cell is equally likely to be chosen by uniform selection.

After a period of search depending on the learning rate, it was discovered that the probability table can prematurely converge on particular genotype-mapping pairings while other locations in the table have no (or practically no) probability associated with them. In order to allow the algorithm to continue to explore all genotype-mapping combinations and the underlying binary sequences they make available to the search space, an additive noise source is introduced (Figure 3.6). This is accomplished by examining each (genotype-associated) column when it is updated to see if any location in that column has exceeded a user-defined threshold γ. If it has, each member of the column has a uniform probability $P(1 - \gamma)$ of having a standard Guassian probability adjustment in the interval [0, 1] added to its current value. (In rarer cases where the Guassian value is outside the interval [0, 1], it is simply re-chosen until it falls in the interval.) The values in the column are then re-normalized so that they sum to unity. For completeness, a summary of the algorithm pseudocode is given in Figure 3.7.

```
Initialize size P mapping & genotype populations
Initialize each value in P x P probTable = 1/P
while (tournamentNotDone && solutionNotFound)
   Use probTable mapping rows for roulette selection of 4
      genotype-mapping pairings
   Rank selectedGenotype & associatedMapping pairings
   Verify or set current bestGenotype & bestMapping
   Update probTable according to Eq. (3.1) & (3.2)
   if (element of winning genotype column > γ)
      for (each column element)
         Add Guassian noise value to element
      Normalize column contents to unity
   for 2 loserGenotypes
      if (mutation threshold is met)
         if (loserGenotype ≠ bestGenotype)
            mutate(loserGenotype = copy of parent)
      if (xover threshold is met)
         if (both loserGenotypes ≠ bestGenotype)
            xover(loserGenotypes = copy of parents)
   for 2 loserMappings
      if (mutation threshold is met)
         if (loserMapping ≠ bestMapping)
            mutate(loserMapping = copy of parent)
      if (xover threshold is met)
         if (both loserMappings ≠ bestMapping)
            xover(loserMappings = copy of parents)
```

Figure 3.7. Pseudocode for the Probabilistic Adaptive Mapping Developmental Genetic Programming (PAM DGP) Algorithm.

3.4 How PAM DGP Addresses the Drawbacks of the Standard Adaptive Mapping Algorithm

PAM DGP addresses the problem of loss of context and fitness spiking (the Red Queen effect) using two components of the algorithm: elitism and the probability table. Firstly, fitness spiking is stopped by protecting the best genotype mapping *combination* (elitism): PAM DGP introduces a small degree of elitism that protects the genotype and mapping in the current best combination from being overwritten, mutated, or crossed over until it is replaced with a new, better genotype/mapping combination. This is accomplished in the algorithm by checking if a losing individual happens to be the best genotype or mapping, and if it is, it does not have its genotype replaced and is not mutated or subject to crossover. The result of this protection of the current best combination is a much more robust algorithm where the evolutionary progress is always retained. This behavior is shown in Figure 3.8 and can be contrasted with the behavior for the Standard Adaptive Mapping algorithm on the same experimental trial shown in Figure 3.2. Dips in best fitness per round occur in PAM DGP as seen Figure 3.8 even though every genotype is present in every round because they can be coupled with any mapping combination, i.e., from the same mapping each to separate mappings for each individual. When the combinations in a round are such that suboptimal mappings are selected across all genotypes, the best fitness in the round dips. However, the individuals in the current highest fitness genotype-mapping pairing are still protected so overall progress is not lost, hence PAM DGP is robust to the fitness dropping (also evidenced by Figure 3.8).

When comparing performance of the algorithms, note that each round of PAM DGP evaluates 4 genotype-mapping pairings as does each round of the Standard Adaptive Mapping implementation. Actually, all algorithms discussed in this thesis use 4 evaluations per round. At any point in this thesis, readers interested in performance based on evaluations need only multiply the tournament round metric by a factor of 4. The difference between the algorithms is simply that each round of the Standard Adaptive Mapping is either a genotype or mapping tournament round (as described in Section 2.3), where there is no such distinction in PAM DGP.

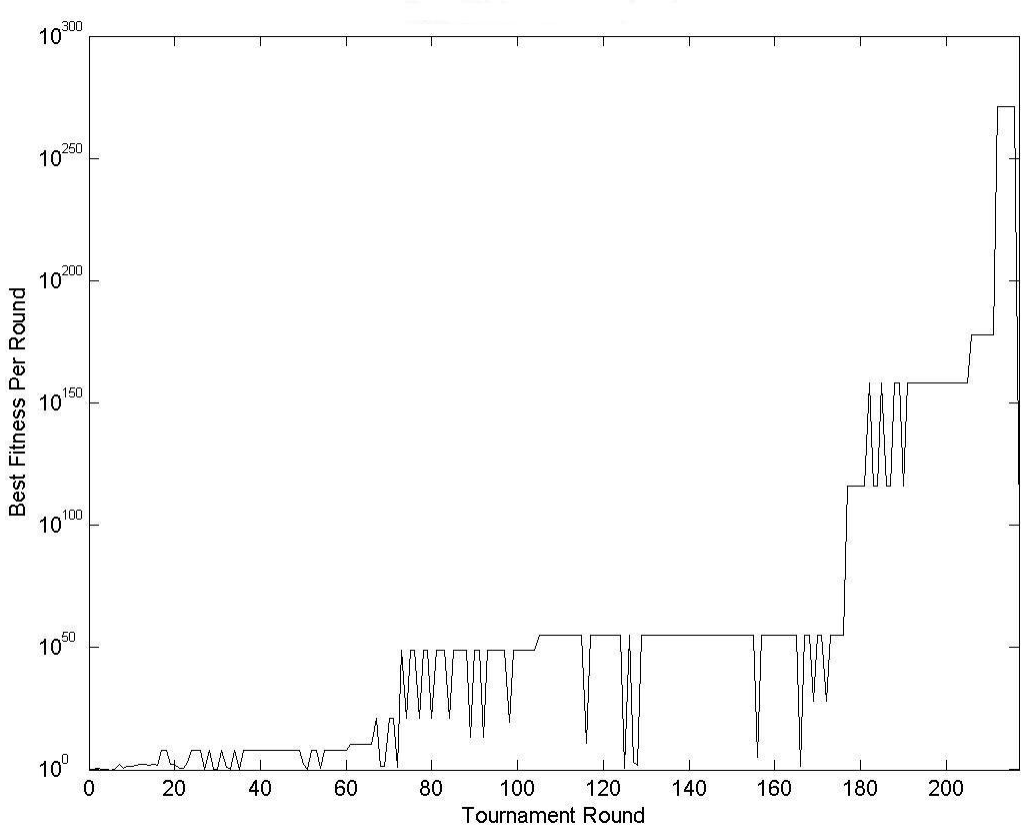


Figure 3.8. Best fitness per round for PAM DGP applied to the MAX Problem. This graph represents a typical run for 200 bit individuals and a population of 8. Breaks in the graph indicate fitness points ≤ 1 on the log scale. Some fitness spikes will still be apparent due to the mapping selection mechanism that avoids purely greedy behaviour.

Loss of context leading to the Red Queen Effect is due to reliance on a single individual (that may change) in one population to set the context for the entirety of the other population. The Red Queen loss of context is minimized by the use of the probability table because the algorithm no longer relies on a single individual from the one population as the fitness context for all the individuals in the second population. This also means that the Standard Adaptive Mapping algorithm suffers from a lack of adequate exploration of the search space by not providing a mechanism for evaluating alternative combinations of genotypes and mappings at multiple stages of evolution. The PAM DGP algorithm features a probability table that guides solution search while allowing the combination of any genotype individual with any mapping during a tournament round, with the selection of the combination made in a fitness-proportionate way. A quantification of the search space considered during a tournament round for each algorithm is provided as a proof in Figure 3.9.

Let n be the number of individuals in either the genotype population or the mapping population.

Let k be a number $< n$ corresponding to a particular individual in either population.

G_i denotes an individual in the genotype population, where $i < n$.

M_i denotes an individual in the mapping population, where $i < n$.

Standard Adaptive Mapping DGP:

For any given mapping round in the Standard Adaptive Mapping Algorithm, the genotype in all combinations produced is fixed (the best genotype).

A combination thus takes the form $(G_k, M_{i\ldots n})$ where G_k has 1 possibility and $M_{i\ldots n}$ has n possibilities.
There are thus 1 x n = n possible combinations for $(G_k, M_{i\ldots n})$.
With 4 combinations selected per round, with no duplication of mappings and one genotype, this gives four combinations:

1. $(G_c, M_{i\ldots n}) \rightarrow 1 \times n = n$ possibilities
2. $(G_c, M_{i\ldots n-1}) \rightarrow 1 \times (n-1) = n-1$ possibilities
3. $(G_c, M_{i\ldots n-2}) \rightarrow 1 \times (n-2) = n-2$ possibilities
4. $(G_c, M_{i\ldots n-3}) \rightarrow 1 \times (n-3) = n-3$ possibilities

Result 1. During each mapping round, then,
$n + (n-1) + (n-2) + (n-3) = 4n-6$ possibilities are considered.

The same argument applies to any genotype round, only substituting G for M.

Probabilistic Adaptive Mapping DGP:

For any given round in the Probabilistic Adaptive Mapping DGP, separate genotype individuals are picked and combined with any mapping.

A combination thus takes the form $(G_{i\ldots n}, M_{i\ldots n})$ where $G_{i\ldots n}$ has n possibilities and $M_{i\ldots n}$ has n possibilities.
There are thus $n \times n = n^2$ possible combinations for $(G_c, M_{i\ldots n})$.
With 4 combinations selected per round, with no duplication of genotypes and duplication permitted for mappings, this gives four combinations:

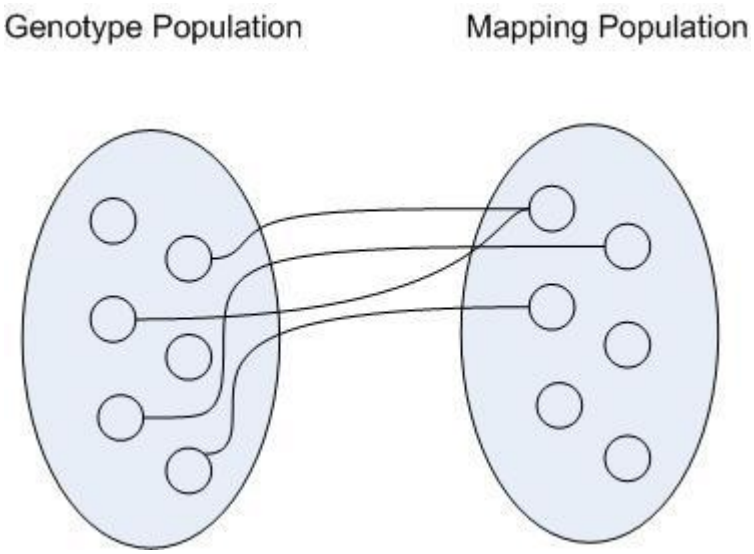
1. $(G_{i\ldots n}, M_{i\ldots n}) \rightarrow n \times n = n^2$ possibilities
2. $(G_{i\ldots n-1}, M_{i\ldots n}) \rightarrow (n-1) \times n = n^2 - n$ possibilities
3. $(G_{i\ldots n-2}, M_{i\ldots n}) \rightarrow (n-2) \times n = n^2 - 2n$ possibilities
4. $(G_{i\ldots n-3}, M_{i\ldots n}) \rightarrow (n-3) \times n = n^2 - 3n$ possibilities

Result 2. During each mapping round, then,
$n^2 + (n^2 - n) + (n^2 - 2n) + (n^2 - 3n) = 4n^2 - 6n = n(4n-6)$
possibilities are considered.

Comparing Result 1 and 2, we see that for mapping and genotype populations of size n, the Probabilistic Adaptive Mapping DGP allows n times as many possible combinations as the standard ($n(4n-6)$ and $4n-6$, respectively) to be considered in any given tournament round.

Figure 3.9. Quantification of the combinations considered by the Standard Adaptive Mapping Algorithm and PAM DGP.

Figure 3.9 demonstrates that the PAM DGP algorithm considers *n* times as many combinations for a population of size *n* as the Adaptive Mapping algorithm during any given tournament round. The computational expense of evaluating an individual's fitness by considering every individual in the alternate population is avoided by using the probability table to provide a guided selection of fruitful combinations, but the solution search can still consider any combination of genotype and mapping, with no duplication of genotypes but duplication of mappings permitted, at any given tournament round.[4] An example of the type of possible combinations that PAM DGP selects in a given tournament round is shown pictorially below in Figure 3.10, which can be contrasted with the selection combinations depicted in Figure 3.3. Recall that there is no distinction between genotype and mapping rounds in PAM DGP.



[4] That is, provided the noise threshold is set to anything less than 1.0, so that no column in the probability table will ever completely exclude any combination by setting the probability of its roulette selection it to 0.

Figure 3.10. One set of possible combinations selected for a tournament of PAM DGP.

To summarize, elitism in PAM DGP allows the algorithm to keep the best genotype and mapping pair that currently generates the best fitness; thus neither the mapping nor the genotype that contribute to the highest fitness can be lost due to changing contexts. The retained top fitness genotype and mapping are not replaced until a better combination appears. The increased exploration of the search space using the probability table allows the exploration of any genotype-mapping combination at any time. This means that a higher fitness combination can be much more readily discovered during the algorithm. Furthermore, the search context for each population is the entirety of the other population throughout the algorithm, preventing the loss of an individual that is central to the context of the other population. PAM DGP thus minimizes an overall lack of fitness-based performance caused by the Red Queen Effect in virtue of its elitism and probability-based selection table. In Chapter 4, we empirically confirm the performance of PAM DGP over the Standard Adaptive Mapping DGP using the Maximum Output benchmark introduced above.

3.5 Computational Complexity Considerations

In addition to increasing exploration of the search space and preventing the loss of context behind the fitness spiking phenomenon of the Standard Adaptive DGP, PAM DGP is no more computationally expensive than the Standard Adaptive Mapping GP. We begin with the complexity analysis of Traditional GP as a benchmark, using the variable n to denote any relevant variable in the algorithms that will scale up with

problem size to produce generalized Big *O* notation. Assume that a GP tournament runs for *n* rounds before the ending or success criterion is met. The evaluation of *n* individuals for each of *n* rounds in Traditional GP only involves execution of some constant number of instructions *n* times, giving linear time $O(n)$ for evaluation of individuals. Following evaluation in each round, we will assume that an efficient sorting mechanism is used in each tournament round for ranking of *n* individuals, such as heap sort which is known to have complexity $O(n \log n)$. Breeding operations (reproduction on *n/2* individuals, crossover and mutation on *n/2* individuals given *n* individuals per tournament round) are then performed in linear time $O(n)$ following ranking. The overall complexity of the Traditional GP can thus be said to be $O(n(n + n \log n + n))$, or, adding and removing constants

$$O(n^2 + n^2 \log n) \in O(n^2 \log n). \tag{3.3}$$

Standard Adaptive Mapping GP involves *n* tournament rounds, where each alternating round involves either a genotype or mapping population round. For each genotype round there is a linear time ($O(n)$) evaluation of *n* genotypes, with an $O(n \log n)$ sort and linear ($O(n)$) breeding time (as described for Traditional GP above). For each mapping round, the evaluation of individuals is no longer in linear time. The evaluation of each mapping individual actually involves execution of Huffman's compression algorithm (only without the usual preliminary step of computation of frequencies for each symbol because the mappings themselves contain that information.) The complexity of Huffman's algorithm used in the Standard Adaptive Mapping GP, implemented as described in [82] and in Section 2.3.1, is computed as follows: The Huffman algorithm essentially creates a single-node tree out of each symbol, giving a set of trees. The two

trees with the smallest probabilities are chosen and combined to form a new tree with a head node specifying the combined probabilities of the nodes, and the two small trees are removed from the set. The Huffman algorithm assigns one frequency node per symbol, with n -1 steps to combine the nodes to build the tree. If a heap is used to store the set of trees as they are combined, the heap can find the largest head node in the set of trees in $log\ n$ time. Tree construction thus takes $O(n\ log\ n)$ time. The symbols are then translated into their Huffman codes, taking $O(n)$ time for n symbols. The overall complexity of the Huffman encoding is thus $O(n + n\ log\ n)$. For the Standard Adaptive Mapping GP, a Huffman encoding occurs for the evaluation of each of n mapping individual during the mapping turn in each round of the tournament. The cost of evaluation of a mapping individual overall is thus $O(n(n + n\ log\ n)) = O(n^2 + n^2\ log\ n)$. The sorting of the evaluated mapping individuals is $O(n\ log\ n)$, and breeding is again $O(n)$. Combining the complexity of the genotype and mapping turns in a n round tournament, the overall complexity of the Standard algorithm of Margetts and Jones is $O(n(n + n\ log\ n + n + n^2 + n^2 log n + n\ log\ n + n))$, or

$$O(n^3 log\ n + n^3 + n^2 log\ n + n^2) \in O(n^3 log\ n). \tag{3.4}$$

PAM DGP involves n tournament rounds of n genotype-mapping pairings rather than separate genotype and mapping turns in each tournament round as in the Standard Adaptive Mapping GP. Each pairing will be evaluated in linear $O(n)$ time by executing the genotype's program to determine fitness based on the genotype program, but the preliminary translation of the n Huffman mapping components for each of the n pairings must take place before the genotype program can be interpreted (even if the mapping individuals need not be explicitly evaluated for fitness as in the Standard Adaptive

Mapping Algorithm). The determination of each of the n mapping encodings will involve Huffman's algorithm using $O(n + n \log n)$ time. Combined, each pairing takes $O(n + n(n + n \log n))$, or $O(n^2 \log n + n^2 + n)$ time to evaluate. There is only one sort of n pairings rather than separate genotype and mapping sorts in each tournament round, giving a complexity of $O(n \log n)$ for heap sort. Breeding of the pairings is done in linear $O(n)$ time. The PAM DGP algorithm also maintains a table of probabilities in memory that is associated with the pairings, and does $n/2$ updates for each of the winning $n/2$ pairings and $n/2$ updates for each of the losing $n/2$ pairings, giving n updates for each of n individuals. The complexity of the table update is thus $O(n^2)$. The overall complexity of PAM DGP using Huffman encodings is thus $O(n\,(n + n + n^2 \log n + n^2 + n + n \log n + n + n^2))$, or

$$O(n^3 \log n + n^3 + n^2 \log n + n^2) \in O(n^3 \log n). \tag{3.5}$$

The computational complexity of the PAM DGP algorithm, when it uses Huffman encoding for the mappings, is thus of the same order as the Standard Adaptive Mapping and no more computationally complex. An alternate direct encoding scheme for the mappings that will be presented in Chapter 6 that will not use the computationally expensive Huffman algorithm with each evaluation, but instead will evaluate individuals in linear $O(n)$ time. The complexity of that algorithm will be shown to be less computationally expensive than the Standard Adaptive Mapping Algorithm (and PAM DGP using Huffman) with complexity of $O(n^3)$ rather than $O(n^3 \log n)$. The relevant components of each algorithm and their associated computational complexities used in the analysis in this Section are summarized below in Figure 3.11.

Traditional GP	Complexity
For *n* rounds	$O(n)$
Evaluate *n* genotypes	$O(n)$
Rank *n* genotypes	$O(n \log n)$
Breed *n* genotypes	$O(n)$

Overall complexity:

$$O(n^2 + n^2 \log n) \in O(n^2 \log n)$$

Standard Adaptive Mapping DGP:	Complexity
For *n* rounds	$O(n)$
Evaluate *n* genotypes	$O(n)$
Rank *n* genotypes	$O(n \log n)$
Breed *n* genotypes	$O(n)$
Evaluate *n* mappings	$O(n)$
Huffman's algorithm	$O(n + n \log n)$
Rank *n* mappings	$O(n \log n)$
Breed *n* mappings	$O(n)$

Overall complexity:

$$O(n^3 \log n + n^3 + n^2 \log n + n^2) \in O(n^3 \log n)$$

PAM DGP using Huffman:	Complexity
For *n* rounds	$O(n)$
Execute mapping encodings	$O(n)$
Huffman's algorithm	$O(n + n \log n)$
Evaluate geno-map pairs	$O(n)$
Rank *n* geno-map pairs	$O(n \log n)$
Breed *n* geno-map pairs	$O(n)$
Update table	$O(n^2)$

Overall complexity:

$$O(n^3 \log n + n^3 + n^2 \log n + n^2) \in O(n^3 \log n)$$

Figure 3.11. Derivation of computational complexities for Traditional GP, Standard Adaptive Mapping GP, and PAM DGP using Huffman-encoded mappings.

Chapter 4. Results using PAM DGP on the MAX Problem

4.1 Problem Definition and Parameterization

As described in the Adaptive Mapping DGP implementation of the MAX problem in Section 3.1, the MAX problem is implemented in PAM DGP with a linear (bit string) stack-based version of GP. Each individual is a first-in first-out (FIFO) stack-based machine composed of a general-purpose stack and an output register. A program that changes the state of the machine is a list of instructions from the function set in Table 4.1, where the program lengths of 50, 100, 150, 200, and 250 bits are tested as in Margetts and Jones [57, 58]. When an instruction in the program is processed sequentially, the required number of arguments is taken from the stack, they are presented to the function, and the return value (if any) is pushed back onto the stack. If there are insufficient arguments on the stack, the function does nothing. The mapping individuals consist of 70 bits, with 10 bits representing a frequency to be associated with each of the seven members of the function set.

In PAM DGP, the dimensions of the probability table are the respective mapping and genotype population sizes. Parameterization of the algorithm thus involves considering a trade off between the quantity of initial genotype and mapping material you want available for the search and the sparseness of the probability grid. What choice will work best depends on the nature of the problem, where we show problems that benefit from choosing either side of the trade-off in this work. The smallest population under which the tournament can still be conducted was found to work best for the simple MAX problem for PAM DGP. PAM DGP and the Adaptive Mapping thus will both initially

use a population of 8 (4 genotypes and 4 mappings). A tournament is stopped when the maximum round limit of 1250 is reached or the success criterion is met. In these experiments, the MAX problem was considered solved when an individual generated a number large enough that it is given the double value of "Infinity" by the Java 2 Runtime Environment, build 1.5.0, on a 1.25 GHz PowerPC G4 running Mac OS X Version 10.4.4 (Tiger). The algorithm was found to work well with a high crossover rate of 0.9, letting the operator do the work of exploring combinations of genetic material to find a solution given smaller starting populations. The mutation operator was found to work well at a conservative rate of 0.1 to prevent disruption of beneficial building blocks. The learning parameter was set to 0.1, indicating that prior search should be emphasized over the latest evaluations. The noise threshold of 0.95 allows the search to very closely search optima without actually being trapped in local optima. All algorithm parameters are summarized below in Table 4.1 for both DGP algorithms (with learning rate and noise threshold only applicable to PAM DGP). As a final remark, general guidelines will be reviewed for establishing PAM DGP parameters in Section 9.1, once the entire PAM DGP model has been presented and empirical evaluation over a wide range of problem domains completed.

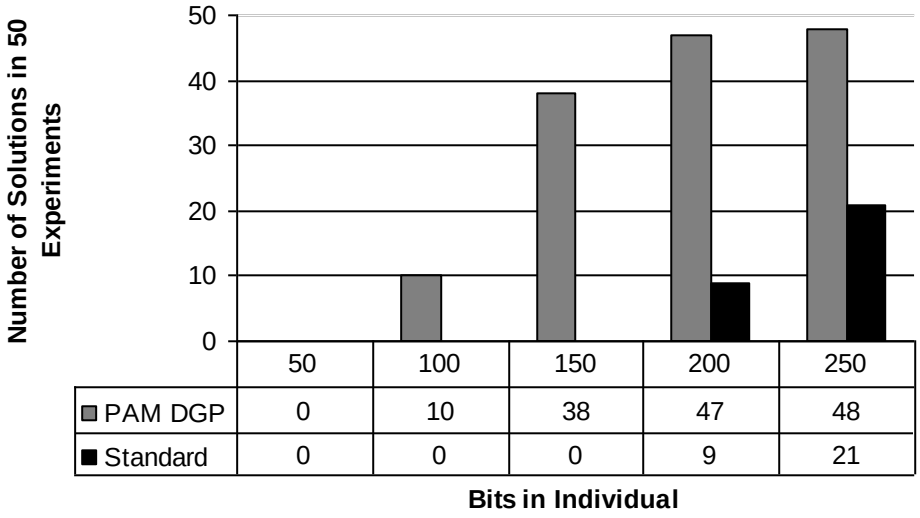
Table 4.1. Maximum Output Problem parameterization.

Tournament Style	Steady State, 4 individuals for each round
Maximum Rounds	1250 (5000 individuals processed)
Experiments	50 independent runs
Function Set	+, *, const, dup, pop, stack2Register, register2Stack
Genotype structure	Stack-based with register; 50, 100, 150, 200, 250 bits
Mapping structure	Adaptive, 70 bits (10 bits per function set member)
Genotype, mapping mutation	Point mutation, threshold = 0.1
Genotype, mapping crossover	Equal-sized blocks, threshold = 0.9
Population size	4 or 25 individuals in each population, traditional population of 50
Fitness	Output register content after evaluation.
Objective	Generate largest number possible.
Termination	Infinity (success) or maximum rounds.
Learning rate	0.1
Noise threshold	0.95

4.2 MAX Problem Results

4.2.1 Results for Equal Population Sizes for Both Implementations

The number of experiments out of 50 independent trials that solved the problem is given below in Figure 4.1, with the mean best fitness for each tournament round over all 50 experiments plotted below in Figures 4.2 to 4.6 for both algorithms using a population of 8 for 50, 100, 150, 200, and 250 bits. Since a successful value of "infinity" is impossible to plot, the fitness measure for any experiment not yet achieving success at a round is used to determine the mean.



	50	100	150	200	250
PAM DGP	0	10	38	47	48
Standard	0	0	0	9	21

Figure 4.1. Number of Maximum Output solutions in 50 independent experiments, given a PAM DGP population of 8 and standard population of 8.

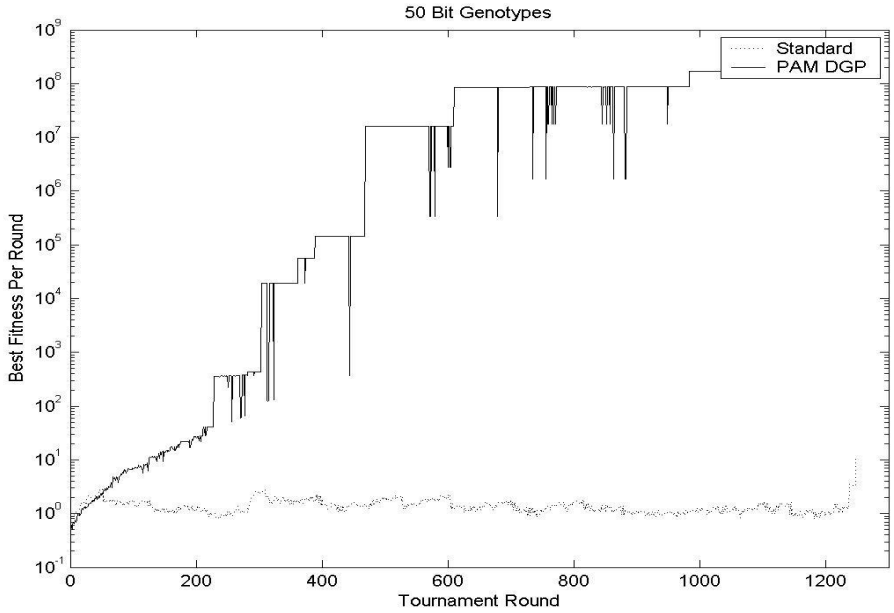


Figure 4.2. Mean best fitness per round on the MAX problem over 50 independent runs for PAM DGP and the Standard Adaptive Mapping algorithm, both with a population of 8, using 50 bit individuals.

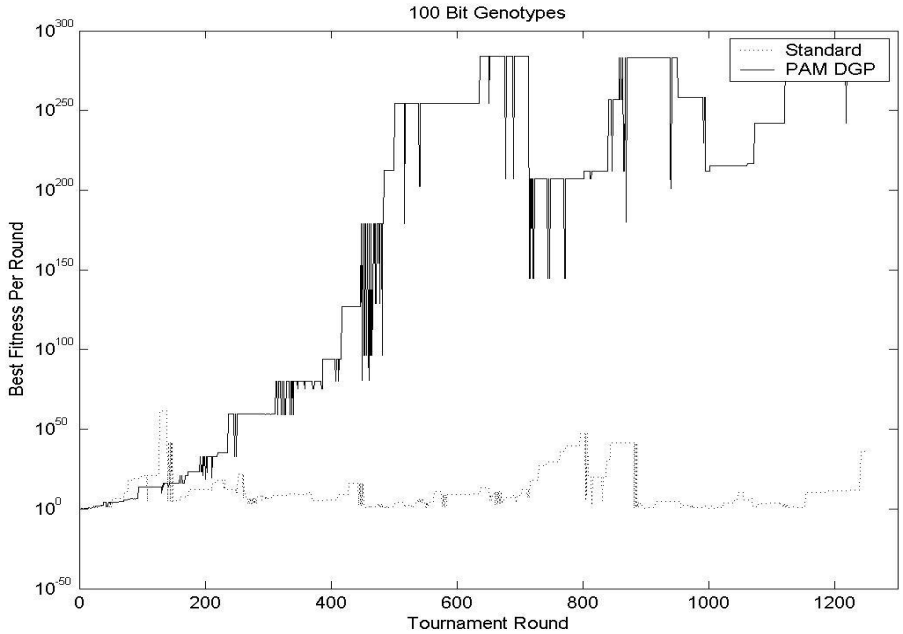


Figure 4.3. Mean best fitness per round on the MAX problem over 50 independent runs for PAM DGP and the Standard Adaptive Mapping algorithm, both with a population of 8, using 100 bit individuals.

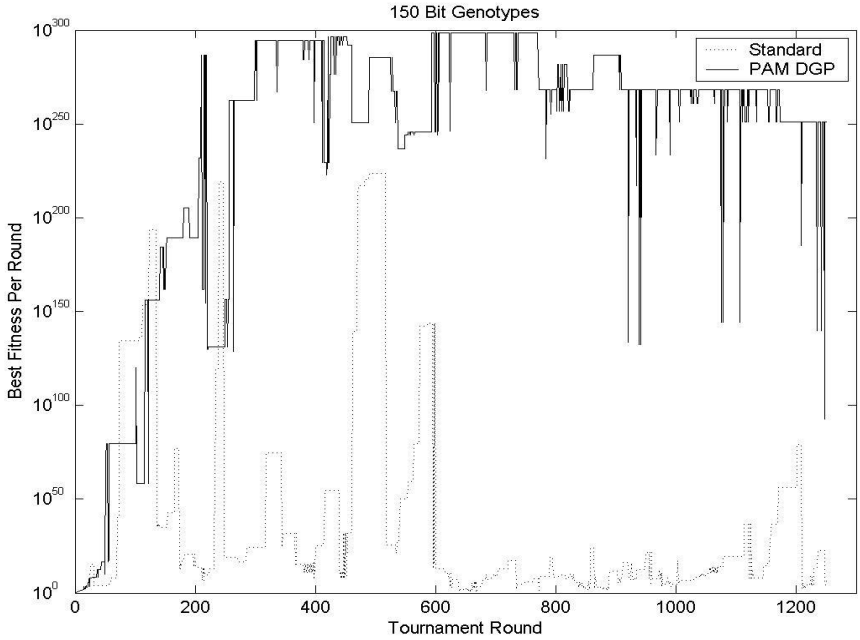


Figure 4.4. Mean best fitness per round on the MAX problem over 50 independent runs for PAM DGP and the Standard Adaptive Mapping algorithm, both with a population of 8, using 150 bit individuals.

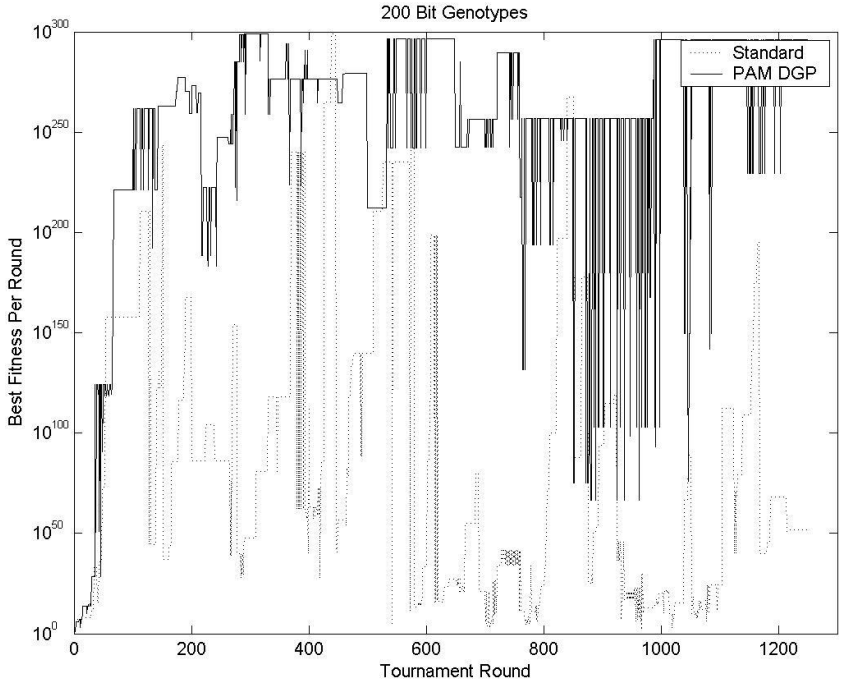


Figure 4.5. Mean best fitness per round on the MAX problem over 50 independent runs for PAM DGP and the Standard Adaptive Mapping algorithm, both with a population of 8, using 200 bit individuals.

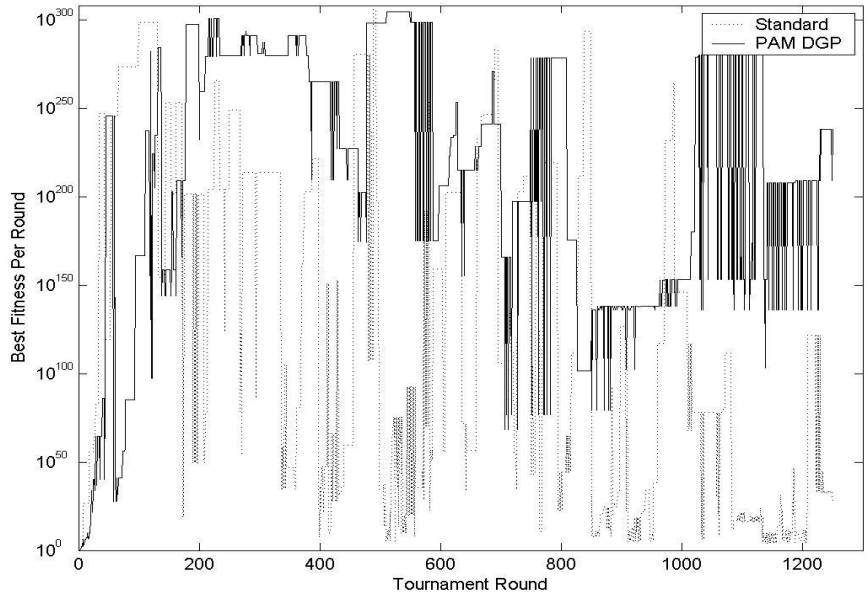


Figure 4.6. Mean best fitness per round on the MAX problem over 50 independent runs for PAM DGP and the Standard Adaptive Mapping algorithm, both with a population of 8, using 250 bit individuals.

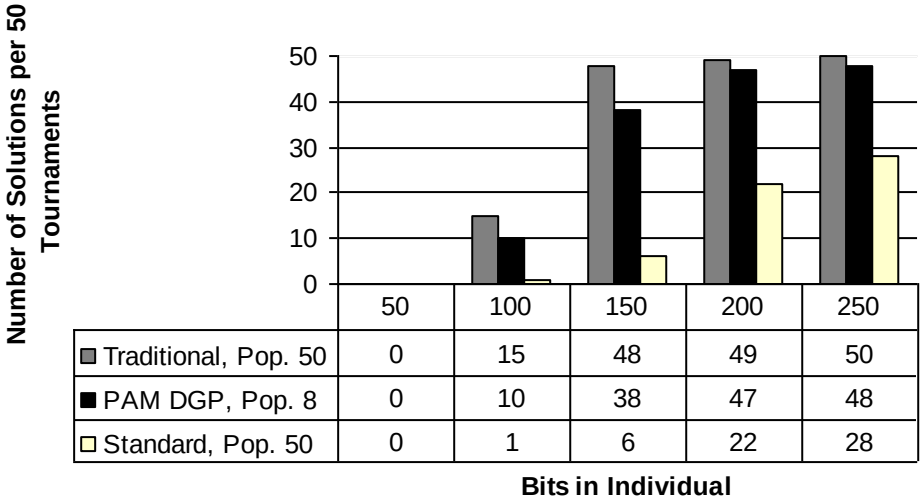
4.2.2 Results for Optimal Population Size for Each Implementation

It is evident from Figure 4.1 and Figures 4.2 to 4.6 that PAM DGP is clearly much better able to generate solutions to the MAX problem and exhibits higher best fitness per round as the algorithm executes. It should be noted that all functions in the function set potentially contribute to the production of larger numbers, that is, there are no detrimental function operators. This means that as the bit length of an individual increases, there is an increased chance of the individual generating a large number. Note that in Figures 4.2 to 4.6, as bit size of individuals increase, the Adaptive Mapping DGP performance approaches PAM DGP across tournament rounds. This is simply a reflection of the problem becoming easier with increasing bit lengths, meaning that the lowest bit length (50 bits) represents the most difficult version of the problem. The fact

that PAM DGP is the clear performance winner in Figure 4.2 is the most meaningful of the plots in Figures 4.2 to 4.6, even though neither algorithm generated solutions for that bit length (Figure 4.1). Naturally, the reflection of increased difficulty with lower bit lengths is also reflected by the number of solutions found by both algorithms in Figure 4.1. However, the general magnitude with which PAM DGP outperforms that Standard Adaptive Mapping DGP across all bit lengths suggests that restricting the Standard Adaptive Mapping DGP to a population of 8 to match PAM DGP may have unfairly hindered its performance.

To achieve fairness for the Standard Adaptive Mapping DGP, it was permitted a starting population of 50 (25 individuals in each population). A population of 50 was found to be a good balance between search space size and genetic material available for search for that algorithm. A Traditional (fixed mapping) GP was also run with a population of 50 individuals for the purpose of comparison. Even with the larger population size, PAM DGP still dramatically outperforms the Standard Adaptive Mapping DGP (see Figure 4.7 below) in terms of number of solutions found. PAM DGP does not outperform traditional GP on this metric, but this is a simple problem where the exploration of the mapping space naturally causes additional search time to be required by PAM DGP. Considering the additional overhead used by PAM DGP, the performance in comparison to Traditional GP is actually rather competitive (compare black and grey bars of Figure 4.7). The MAX problem is used by Margetts as the benchmark when introducing the Standard Adaptive Mapping algorithm; the main focus of this Section is to demonstrate the performance advantages of PAM DGP over the Standard Adaptive Mapping DGP, not yet necessarily outperforming Traditional GP. Traditional GP is thus

provided in this section (when pertinent) because it is an expected benchmark comparison.



	50	100	150	200	250
Traditional, Pop. 50	0	15	48	49	50
PAM DGP, Pop. 8	0	10	38	47	48
Standard, Pop. 50	0	1	6	22	28

Figure 4.7. Number of Maximum Output solutions in 50 independent experiments, given a PAM DGP population of 8, Standard Adaptive Mapping population of 50, and Traditional population of 50.

The mean best fitness for each tournament round over all 50 experiments is plotted below in Figures 4.8 to 4.12 for both algorithms using their respective optimal populations for 50, 100, 150, 200, and 250 bits. The PAM DGP algorithm (solid line) outperforms the Standard Adaptive Mapping DGP consistently throughout all tournament rounds for all bits levels. The algorithm is also more robust, as far fewer fitness spikes are evident in the PAM DGP trend line.

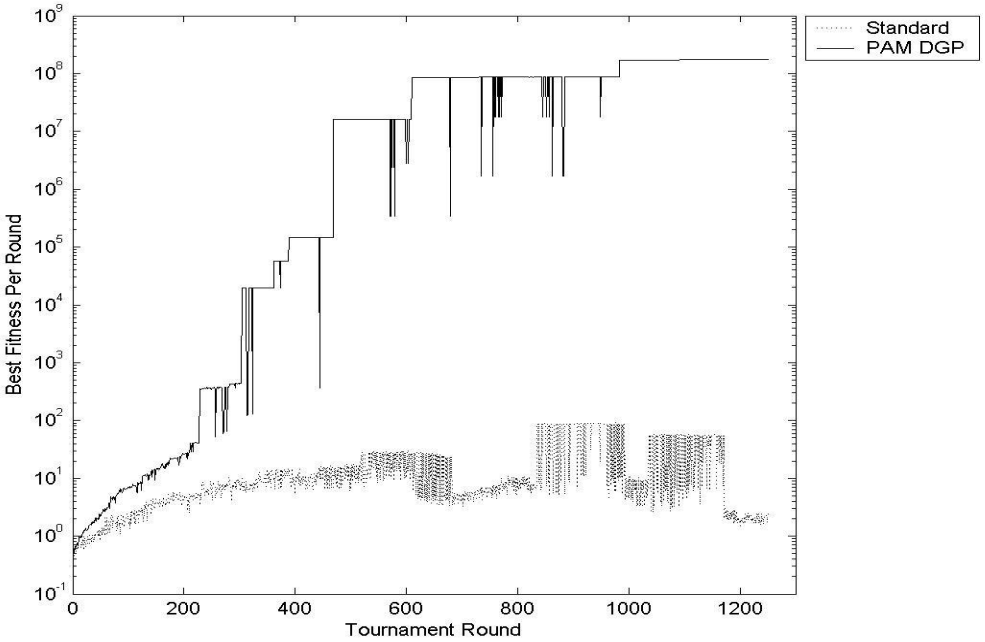


Figure 4.8. Mean best fitness per round over 50 independent runs for PAM DGP, population of 8, and the Standard Adaptive Mapping algorithm, population of 50, on the MAX problem, 50 bit individuals.

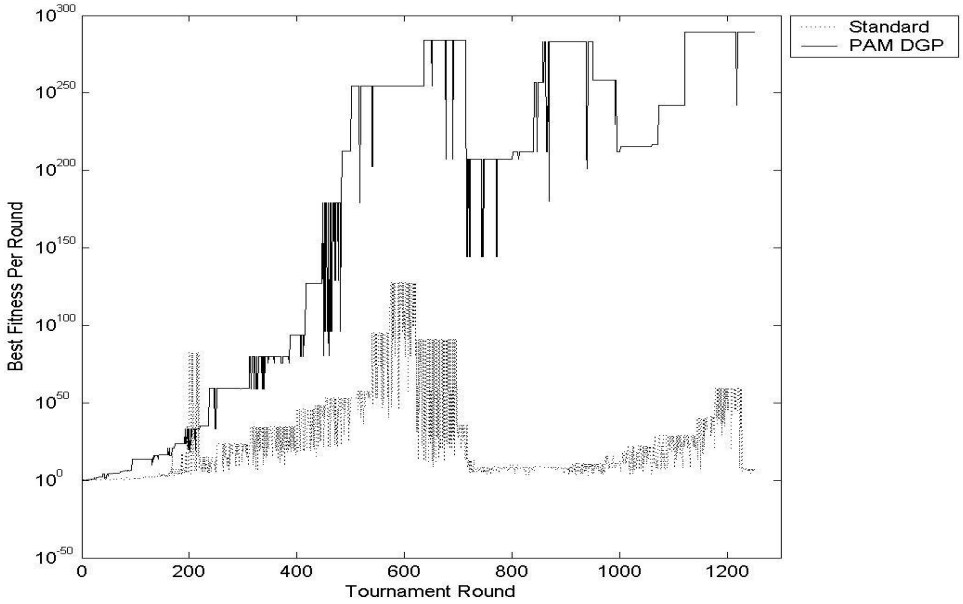


Figure 4.9. Mean best fitness per round over 50 independent runs for PAM DGP, population of 8, and the Standard Adaptive Mapping algorithm, population of 50, on the MAX problem, 100 bit individuals.

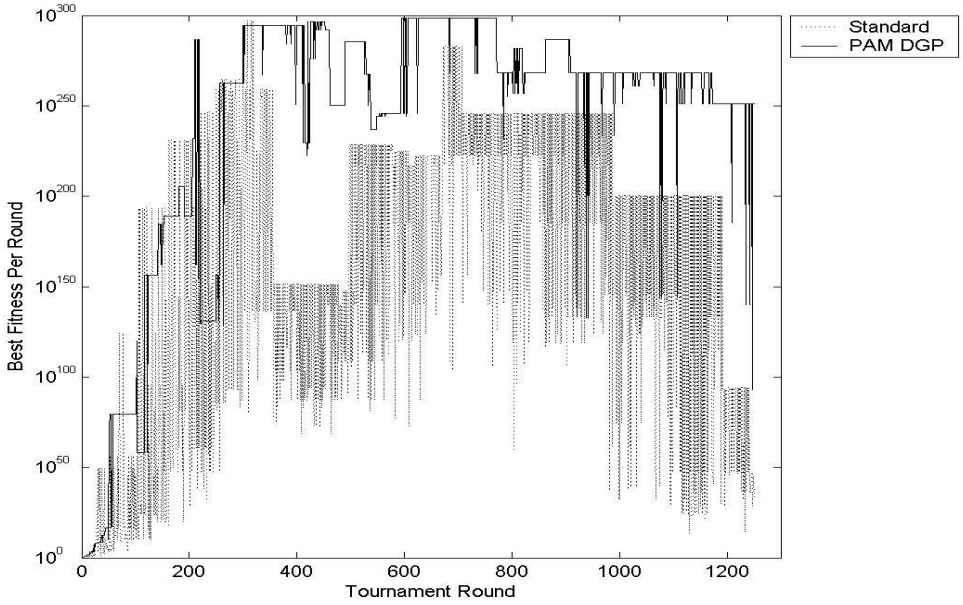


Figure 4.10. Mean best fitness per round over 50 independent runs for PAM DGP, population of 8, and the Standard Adaptive Mapping algorithm, population of 50, on the MAX problem, 150 bit individuals.

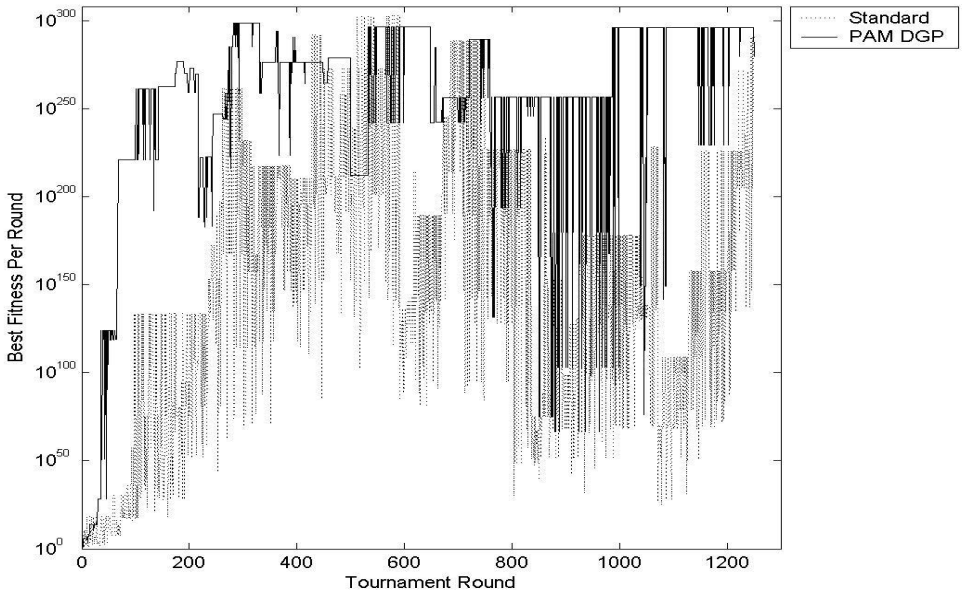


Figure 4.11. Mean best fitness per round over 50 independent runs for PAM DGP, population of 8, and the Standard Adaptive Mapping algorithm, population of 50, on the MAX problem, 200 bit individuals.

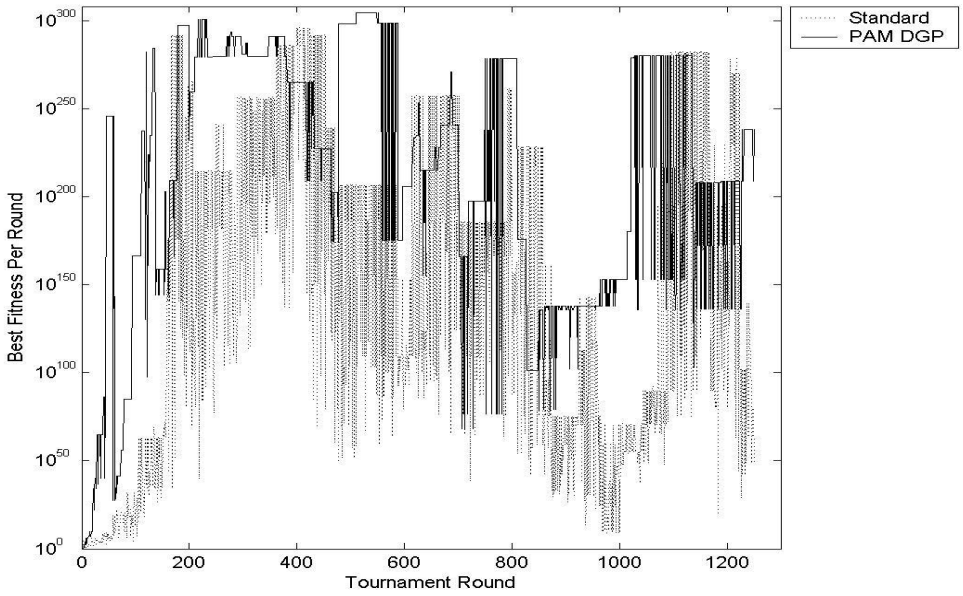


Figure 4.12. Mean best fitness per round over 50 independent runs for PAM DGP, population of 8, and the Standard Adaptive Mapping algorithm, population of 50, on the MAX problem, 250 bit individuals.

The operator content of the optimal population solutions as a percentage of total operators is shown in Figure 4.13 along with p-values to indicate acceptance or rejection of the hypothesis that the symbol frequencies in solutions of the PAM DGP and Standard Adaptive Mapping DGP are equal. To be as stringent as possible with error bar estimates, each bar corresponds to a 95% confidence interval (CI) using the two tailed t-distribution rather than the standard uniform distribution. There is no significant difference at the 0.95 confidence intervals for 5 out of 7 operators, where PAM DGP used less constants and more multiplication than the Standard Adaptive Mapping (both significant at the 99% CI). Since the best solutions only require repeated duplication/multiplication following initial presence of a constant, the difference in symbols would only give PAM DGP more ideal solutions than the Standard. Traditional GP only differs from PAM DGP at the 0.95 CI with respect to TIMES and PLUS, with

Traditional GP providing a greater emphasis on TIMES than the other algorithms. While TIMES is a good operator to emphasize, the Traditional GP does not manage to place as much emphasis on the equally important operator DUP as on TIMES, while the mapping algorithms allow for a more balanced allocation of DUP to TIMES for effective repeated squaring.

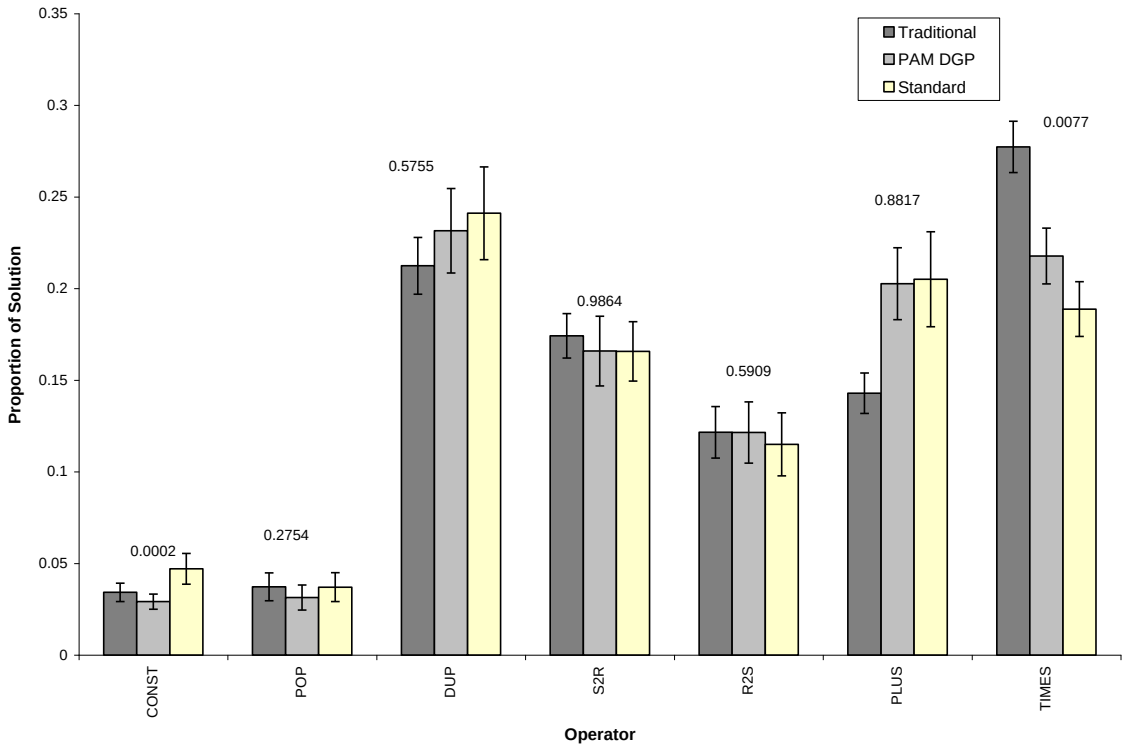


Figure 4.13. Mean operators as proportion of total solution for 200 bit, PAM DGP, population 8, and Standard Adaptive Mapping DGP and Traditional GP, population 50, MAX Problem over 50 trials. P-values for the comparison of PAM DGP and Standard Adaptive Mapping algorithms are displayed above data for each operator. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval.

A final performance metric to be considered is how quickly each algorithm found a solution (or how efficient they were). Figure 4.14 below shows the final number of

tournament rounds (either due to success or reaching maximum rounds) for all bit lengths. As before, each bar corresponds to a 95% confidence interval using the two-tailed t-distribution rather than the standard uniform distribution. PAM-DGP is definitively more efficient than the Adaptive Mapping DGP for bit lengths 150 to 250 at the 0.99 CI, and in no case is outperformed by the standard. PAM DGP is also competitive with Traditional GP, but there is no statistically significant difference at the 0.95 confidence interval between Traditional GP and PAM DGP in the most constrained (difficult) versions of the problem (50-200 bits). The easiest version (250 bit) has a p-value of 0.0203 (not shown) with respect to PAM DGP and Traditional GP.

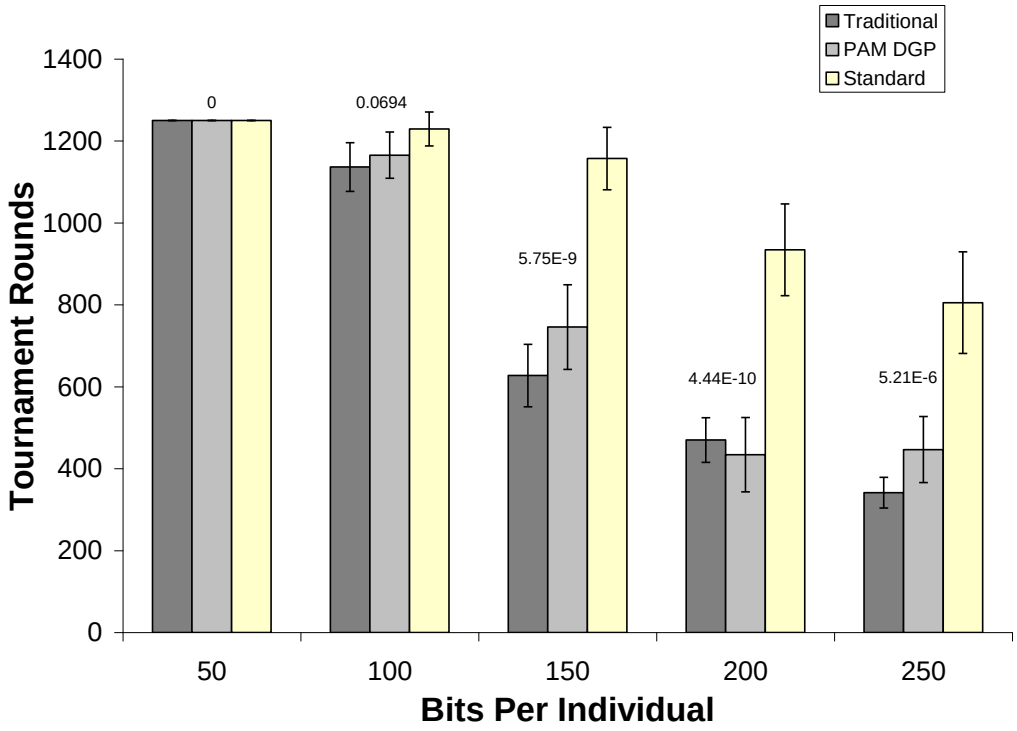


Figure 4.14. Mean number of tournament rounds (reaching maximum rounds or a solution) for PAM DGP, population 8, and Standard Adaptive Mapping DGP, and Traditional GP, population 50, in 50 trials for the Maximum Output problem. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. Corresponding p-values with respect to Standard Adaptive Mapping and PAM DGP are displayed above each pair of data points.

4.3 MAX Problem Summary and Discussion

In this chapter, the Standard Adaptive Mapping and PAM DGP were compared on the Maximum Output problem. PAM DGP was found to solve this simple problem well with the smallest population required to conduct the steady state tournament (4 genotypes and 4 mappings). PAM DGP, however, was found to considerably outperform the Adaptive Mapping DGP at this population level, so the Adaptive Mapping DGP was permitted a population of 50 individuals that was found to be approximately optimal. Based on respective optimal populations, PAM DGP was found to generate more solutions and exhibit best fitness per round trends that outperformed the Standard Adaptive Mapping DGP. Furthermore, it was found to be competitive with Traditional GP but was unable to outperform it on this simple artificial problem

Chapter 5. Results using PAM DGP on Harder Regression Problems

In this chapter, we examine the performance of PAM DGP compared to the Standard Adaptive Mapping DGP for regression problems that are harder than the MAX problem: the Two Boxes problem and the Hénon Mapping. Both problems are harder than the MAX problem in that function sets of both problems involve operators that are extraneous to the problem. That is, not all symbols in the function set are required in an ideal solution. Also, in contrast to the MAX problem, some members of the function set, if not used properly, can be detrimental to producing a solution. Aside from function set considerations, both problems present a considerably more complex optimization goal than simply generating the largest number possible: In both cases, a reasonable approximation to a function is to be produced as a solution and neither function is considered trivial for GP search.

5.1 Two Boxes Problem

5.1.1 Problem Definition and Parameterization

The Two Boxes problem is to relate six independent variables (L_0, W_0, H_0, L_1, W_1, and H_1) through the equation for the difference in volume of two boxes:

$$L_0 W_0 H_0 - L_1 W_1 H_1 \tag{5.1}$$

Although of an artificial nature, the problem has repeatedly been demonstrated to be a challenging problem. The problem is structured as described by Koza in [49] with ten fitness cases that are created using uniform sampling of integers over the interval [1, …,

10]. Fitness is measured as the summed absolute error over all fitness cases, where in each case an absolute error ≤ 0.01 counts as a hit. The success criterion is to produce 10 hits. Four function operators were used: addition (+), subtraction (-), multiplication (*), and division protected against underflow and overflow (%). Only multiplication and subtraction (Equation 5.1) are required for the ideal solution.

The algorithm parameterization is changed from the previous problem to better suit a considerably harder regression problem, with the salient differences being a separation of genotype/mapping crossover rates and mutation type and rates. The crossover rate in mappings is lowered to provide a more stable mapping background against which the genotypes could evolve. Increased mutation rates are used in the genotypes to allow exploration against the backdrop of the more persistent (due to lower operator rates) mappings. Point mutation at a rate of 0.1 is still used in the mappings to ensure a number of frequencies per operator are explored conservatively; whereas an instruction-level XOR mutation operator (the instruction chosen with uniform probability) is now used for the genotype mutation operator to allow enhanced exploration through introduction of additional new genotype material than the point mutation alternative. Furthermore, the genotype mutation operator's rate is raised from 0.1 to 0.5 to expedite exploration of genotype material. Both algorithms perform best with a population of 50 individuals (25 genotypes and 25 mappings) in this problem, benefiting from the additional genetic material with which to perform a search (although we also report results for the minimalist case of a population size of 8). Parameterization for the Two Boxes problem is summarized in Table 5.1 below.

Table 5.1. Two Boxes Problem parameterization.

Tournament Style	Steady State, 4 individuals
Maximum Rounds	50 000 (initial experiments), 150 000
Experiments	50 independent runs
Function Set	+, *, -, % (protected against underflow/overflow)
Terminal Set	L_0, W_0, H_0, L_1, W_1, H_1
Genotype structure	Instruction sequence with 4 registers; 320 bits
Mapping structure	Adaptive, 40 bits (10 bits per function set member)
Genotype mutation	XOR mutation, threshold = 0.5
Mapping mutation	Point mutation, threshold = 0.1
Genotype crossover	Equal-sized blocks, threshold = 0.9
Mapping crossover	Equal-sized blocks, threshold = 0.1
Popultation size	4 or 25 individuals in each population (traditional population of 50)
Fitness Cases	10 sets of 6 integers in [1, …, 10], output value
Fitness	Summed absolute error.
Objective	Fit equation to $L_0W_0H_0 - L_1W_1H_1$
Hits	Number of fitness cases with absolute error ≤ 0.01
Termination	10 hits (success) or maximum rounds.
Learning rate	0.1
Noise threshold	0.95

5.1.2 Interpretation of Instructions

An explanation of instruction interpretation was not required for the MAX problem, as each instruction was a single command with a bit sequence entirely dictated by the Huffman encoding. The command always operated on whatever happened to be included in the individual's stack, with the function not performing any operation when there were insufficient operators on the stack. In contrast, the Two Boxes problem individuals include fixed registers and there are problem fitness case elements that must be referenced by an individual's instructions, so a scheme for interpreting the bit content of the instructions is required. The instruction interpretation is depicted below in Figure 5.1, and is described in the rest of this section.

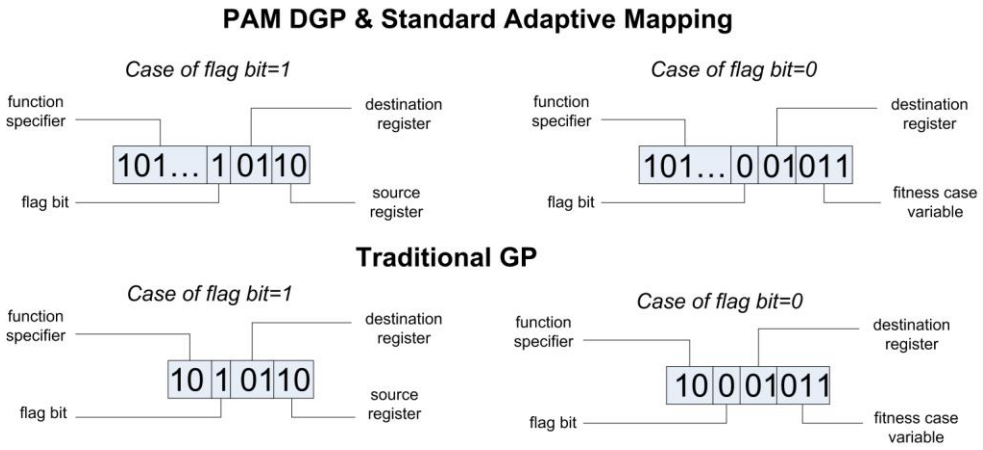


Figure 5.1. Interpretation of instructions for the Two Boxes problem.

Each genotype individual in the Two Boxes problem consists of up to 320 bits and 4 registers. The first segment of an instruction in any algorithm involving Huffman-encoded adaptive mappings (Standard Adaptive Mapping DGP or PAM DGP) identifies one of the possible function operators where the length of this function identifier is dictated by Huffman's algorithm. More emphasis is generally placed on an operator by having a shorter identifier. In Traditional GP, the function identification segment of the instruction is of a fixed length (2 bits), where the integer representation of the bits in this segment is simply used to specify one of the four operators. A single 'flag' bit following the function identifier determines where the operand to the right of the operator is loaded from: if the flag bit is set to '1,' the operand is loaded from the registers, and if the flag bit is set to '0,' the operand is loaded from one of the six variables from the current fitness case. In the case where the operand is loaded from the registers, the integer representation of the two bits following the flag bit specify one of the 4 registers as the destination register, and the last two bits specify one of 4 registers as the source register. In the case where the operand is loaded from the current fitness case, the two bits

following the flag bit specify one of the 4 registers as the destination register, and the three bits following the flag bit specify the one of the six variables from the current fitness case as the operand (where the binary representations of the integers wrap back to the first attribute after the last attribute). All instructions are initialized to random binary sequences and all registers are initially set to '1'.

5.1.3 Two Boxes Results

Neither algorithm provided a solution with a population of 8, with only PAM DGP providing one solution during 50 trials with a population of 50 based on 50 initial trials of 50 000 rounds each. From these initial results, it was evident that for this harder regression problem the larger population of 50 would be beneficial to provide initial genetic material on which to perform solution search. Fitness performance for both the Standard Adaptive Mapping and PAM DGP were thus compared for both starting populations to determine if the larger population would reduce the performance of PAM DGP when compared to the Standard Adaptive Mapping algorithm. Mean best fitness produced over 50 trials is shown below in Figure 5.2. For both starting populations of 8 and 50, PAM DGP outperformed the Adaptive Mapping algorithm (lower fitness values are better). To determine whether providing PAM DGP with a larger starting population led to any performance loss for the PAM DGP algorithm itself, a t-test was performed to compare the two populations in PAM DGP (dark grey bars). The result yielded a p-value of 0.331, indicating that there is no statistically significant difference between the fitness of the two algorithms at the 95% confidence interval. It could thus be expected that overall fitness performance of PAM DGP would not be affected by raising population

size to handle harder problems with more obscure solutions, such as the Two Boxes problem.

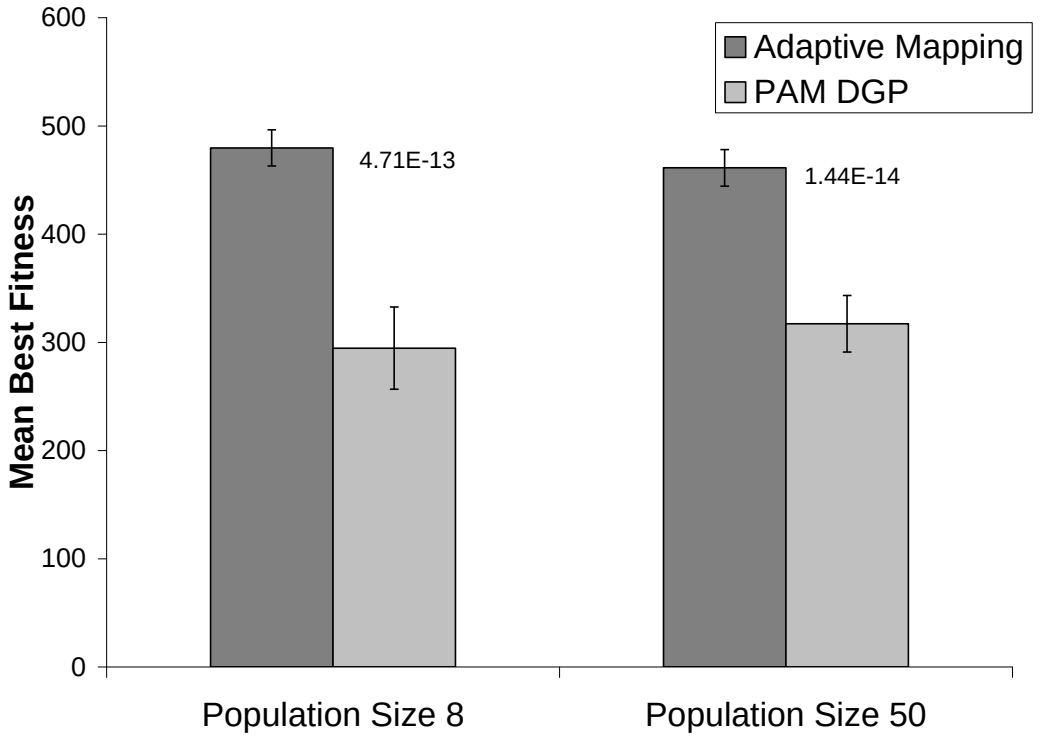


Figure 5.2. Mean best fitness achieved (after maximum rounds or a solution) for PAM DGP and Standard Adaptive Mapping algorithm for 50 trials for the Two Boxes problem. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. Corresponding p-values are displayed above each pair of data points.

Once it was established that increasing population size was not likely to affect performance for PAM DGP on interesting problems, additional tournaments of 150 000 rounds were run to provide the algorithms with a longer tournament count (more evolutionary steps) in which to solve the problem and provide better analysis. The mean best fitness achieved for Traditional GP, PAM DGP, and the Standard Adaptive Mapping DGP is shown below in Figure 5.3. PAM DGP and Traditional GP clearly outperform

the Standard Adaptive Mapping, with Traditional GP just outperforming PAM DGP. Considering the additional overhead of mapping search in the PAM DGP algorithm, its performance can be considered competitive with Traditional GP (especially considering the PAM DGP generated more actual solutions that Traditional GP).

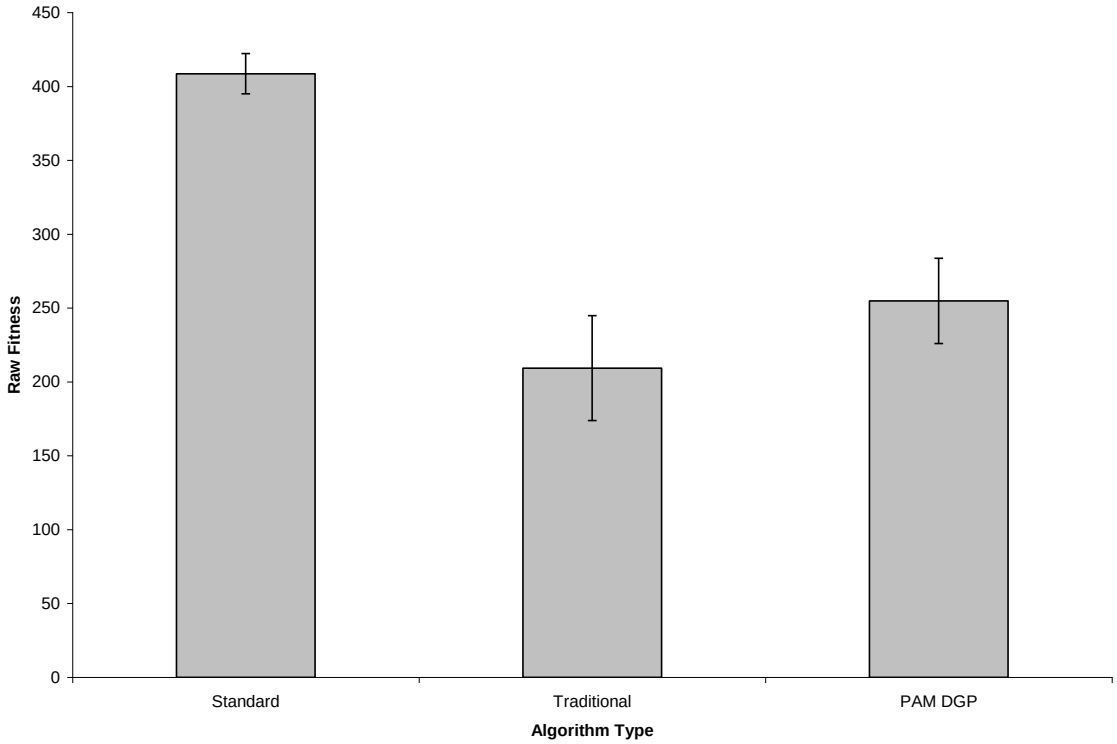


Figure 5.3. Mean best fitness achieved (after maximum rounds or a solution) for PAM DGP, Standard Adaptive Mapping DGP, and the Traditional GP algorithm for 50 trials for the Two Boxes problem. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval.

The final fitness measure for this problem is a useful, but not necessarily most informative, performance measure. Investigation of the problem solution files show that all of the algorithms can either approach the solution gradually in terms of best fitness or can solve the problem suddenly after jumping from a lower fitness. Table 5.2 below provides the number of solutions found in 50 trials of 150 000 rounds each, and how many tournament rounds were required to reach the solution for each algorithm. PAM

DGP finds more solutions to the problem over 50 trials, and has the fastest time to solution (36 161 rounds), despite the additional overhead of mapping exploration in addition to the exploration of the genotype-based search space.

Table 5.2. Two Boxes solutions for Traditional GP, Standard Adaptive Mapping and PAM DGP.

Algorithm	Solutions per 50 Trials	Round Solution Found
Traditional GP	1	42004
Standard Adaptive Mapping	0	n/a
PAM DGP	2	134743, 36161

The operator content of the solutions as a percentage of total operators is shown in Figure 5.4 below along with p-values for the hypothesis that the symbol frequencies of the two algorithms are the same. There is no significant difference at the 0.95 confidence interval for all operators, indicating that both algorithms were fairly similar in their ability to choose useful function set symbols and that overall the algorithms chose similar solution content. Traditional GP, on the other hand, selected different numbers of operators from PAM DGP at the 0.95 confidence interval for MULT, ADD, and DIVIDE. Namely, the proposed Traditional GP ‘solutions’ feature more addition and less multiplication, which is not an effective trade-off for this problem.

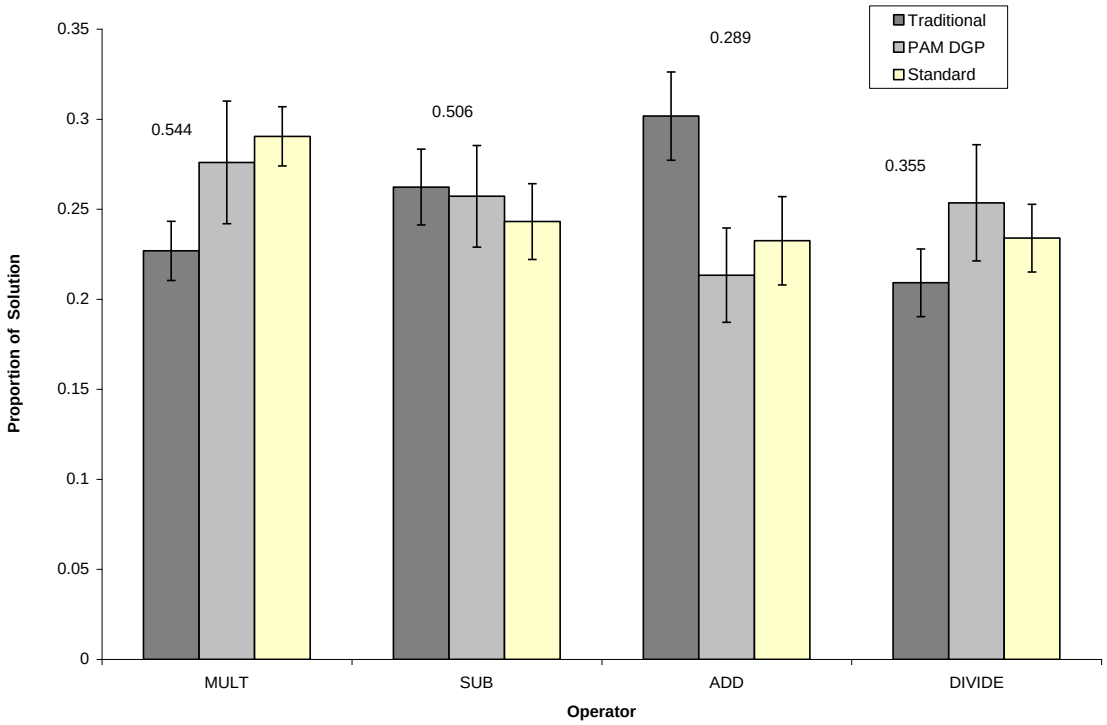


Figure 5.4. Mean operators as percentage of total solution over 50 trials for the Two Boxes Problem using PAM DGP, Standard Adaptive Mapping algorithm, and Traditional GP. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. Corresponding p-values are displayed above each pair of data points for PAM DGP and the Standard Adaptive Mapping DGP.

5.2 The Hénon Map

5.2.1 Problem Definition and Parameterization

Our third regression problem is the astronomer Hénon's famous equation that represented an attempt to model the orbits of stars around a galactic centre [30] which produced a time series resulting in a chaotic attractor. The chaotic time series means that the value of each consecutive result of the equation appears in a disorderly manner, but the two and three dimensional plots of the equation produce an orderly curve through space (a chaotic attractor). The equation is expressed as

$$x_t = 1 - a(x_{t-1})^2 + b(x_{t-2}) \tag{5.2}$$

where $x_0 = 0$, $x_1 = 0$, $a = 1.4$, and $b = 0.3$. A two-dimensional and three-dimensional plot of the chaotic attractor produced by the 2000 points of the time series is shown below in Figure 5.5. The first 100 points of the Hénon time series x_t against t are plotted in Figure 5.6.

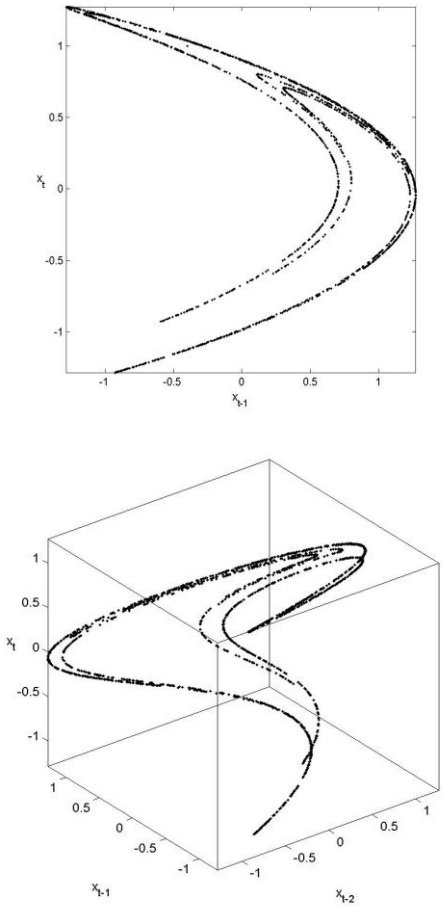


Figure 5.5. The Hénon Map plotted in two (top) and three (bottom) dimensions.

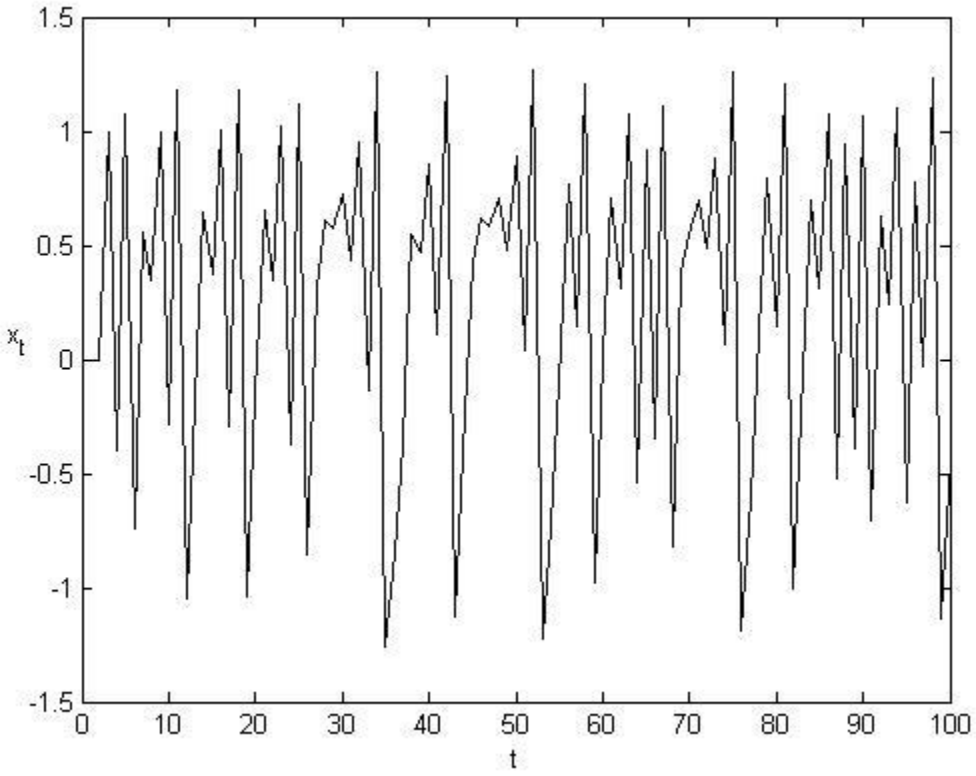


Figure 5.6. First 100 points (t, x_t) in the Hénon time series.

Margetts [57] used the Adaptive Mapping algorithm with a stack-based structure of genotype individuals on this problem to extract an equation based on 500 fitness cases using 2 successive times as inputs and the result of Equation 5.2 as the output. No significant improvement over traditional GP (global, fixed mapping) performance was found in his thesis based on current maximum fitness achieved at each round.

To compare PAM DGP's performance to the Adaptive Mapping algorithm, we use a genotype individual structure similar to that which was successful for the Two Boxes Problem. That is, each individual consists of 4 registers and a binary encoded program. The setup of Margetts's experiment in [57, 58] was replicated: 500 fitness cases were used, and each experiment was run for 10 000 fitness evaluations (or 2500

rounds given 4 fitness evaluations per round). Each fitness case used x_{t-2} and x_{t-1} from Equation 5.2 as inputs and x_t as the output. The aim of the experiment is similar to the goal of Two Boxes; that is, generate the original formula using the series of data points. Fitness was measured as the summed squared error over all fitness cases, where in each case an absolute error ≤ 0.01 counted as a hit. The success criterion was to produce 500 hits (a hit for every fitness case). Eight functions were used for this problem: addition (+), subtraction (-), multiplication (*), load constant (CONST), protected division (%), square (SQUARE), square root (SQRT), and no operation (NOOP). Hénon's equation only requires the first four of these operators; the others are added to make the problem more difficult. Emphasis of functional operators is again achieved by the Huffman mapping varying bit lengths and combinations used to start each instruction. The parameterization is similar to the Two Boxes problem in relation to mutation and crossover rates and types to suit a harder regression problem, and a population of 50 was used to provide adequate genetic material for the search. The parameter values used are shown in Table 5.3 below.

Table 5.3. Hénon Mapping Problem parameterization.

Tournament Style	Steady State, 4 individuals for each round
Maximum Rounds	2 500
Experiments	50 independent runs
Function Set	+, *, -, CONST, % (protected), SQUARE, SQRT, NOOP
Terminal Set	x_{t-1}, x_{t-2}
Genotype structure	Instruction sequence with 4 registers; 320 bits
Mapping structure	Adaptive, 80 bits (10 bits per function set member)
Genotype mutation	XOR mutation, threshold = 0.5
Mapping mutation	Point mutation, threshold = 0.1
Genotype crossover	Equal-sized blocks, threshold = 0.9
Mapping crossover	Equal-sized blocks, threshold = 0.1
Population size	25 individuals in each population (50 for traditional)
Fitness Cases	500 sets of 2 values in time series with x_0=0, x_1=0, and output value
Fitness	Summed squared error.
Objective	Fit equation to $x_t = 1 - 1.4(x_{t-1})^2 + 0.3(x_{t-2})$
Hits	Number of fitness cases with absolute error ≤ 0.01
Termination	500 hits (success) or maximum rounds
Learning rate	0.1
Noise threshold	0.95

5.2.2 Interpretation of Instructions

Each genotype individual in the Hénon problem consists of up to 320 bits and 4 registers. The first segment of an instruction identifies one of the eight possible operators, where the length of this function identifier is dictated by the encoding it is allotted by Huffman's algorithm. More emphasis is generally placed on an operator by having a shorter identifier. In Traditional GP, the function identification segment of the instruction is of a fixed length (3 bits), where the integer representation of the bits in this segment is simply used to specify one of the eight operators.

For the operators { +, -, *, %, SQUARE, SQRT}, a single ‘flag’ bit following the function identifier determines where the operand to the right of the operator is loaded from: if the flag bit is set to ‘1,’ the operand is loaded from the registers, and if the flag bit is set to ‘0,’ the operand is loaded from one of the five variables from the current fitness case. In the case where the operand is loaded from the registers, the integer representation of the two bits following the flag bit specify one of the 4 registers as the destination register, and the last two bits specify one of 4 registers as the source register. In the case where the operand is loaded from the current fitness case, the two bits following the flag bit specify one of the 4 registers as the destination register, and the bit following the destination register specification identifies one of the two variables (x_{t-1} or x_{t-2}) from the current fitness case as the operand. The exceptions to this parsing are the CONST (constant) function and NOOP (no operation) functions. For the CONST operator, following the interpretation of the functional bits, the next 16 bits in the instruction are translated into a decimal in the range [0, 1] (as was the case for Margetts [57, 58]). The NOOP operator simply performs no action; only the functional bits are interpreted and nothing more. The final answer following the execution of the instructions is located in the first register. All instructions are initialized to random binary sequences and all registers are initially set to ‘1’. Interpretation of an instruction for the Hénon problem is shown below in Figure 5.7.

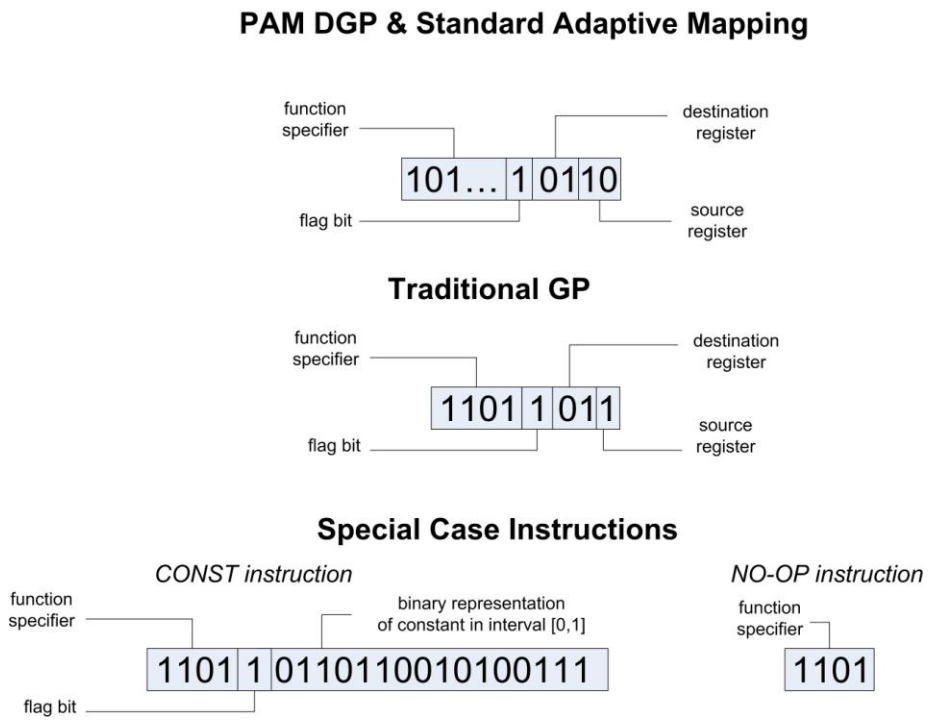


Figure 5.7. Interpretation of instructions for the Hénon problem.

5.2.3 Hénon Mapping Results

No algorithms were able to solve the problem in all of the 50 independent trials, although low mean squared error rates indicate that reasonable approximations to the Hénon mapping were produced. Margetts also did not achieve a solution, but did produce good approximations in [57]. Mean best fitness among individuals competing in each tournament round (restricted to the interval [0.35, 1.0] for clarity) is shown below in Figure 5.8 to compare the two mapping algorithms. PAM DGP began the tournament with higher mean squared error rates, but was exploring solutions with lower error rates than the Standard Adaptive Mapping DGP within the final half of the tournament. This can be noted by both a general trend towards low cumulative error rates in the PAM DGP graph from mid-tournament onwards. The Adaptive Mapping DGP, in contrast, actually

lost search progress it had made in earlier tournament rounds in the later rounds. Adaptive Mapping DGP also exhibited considerable spiking into regions of higher error levels indicating frequent loss of progress in the solution search throughout the tournament.

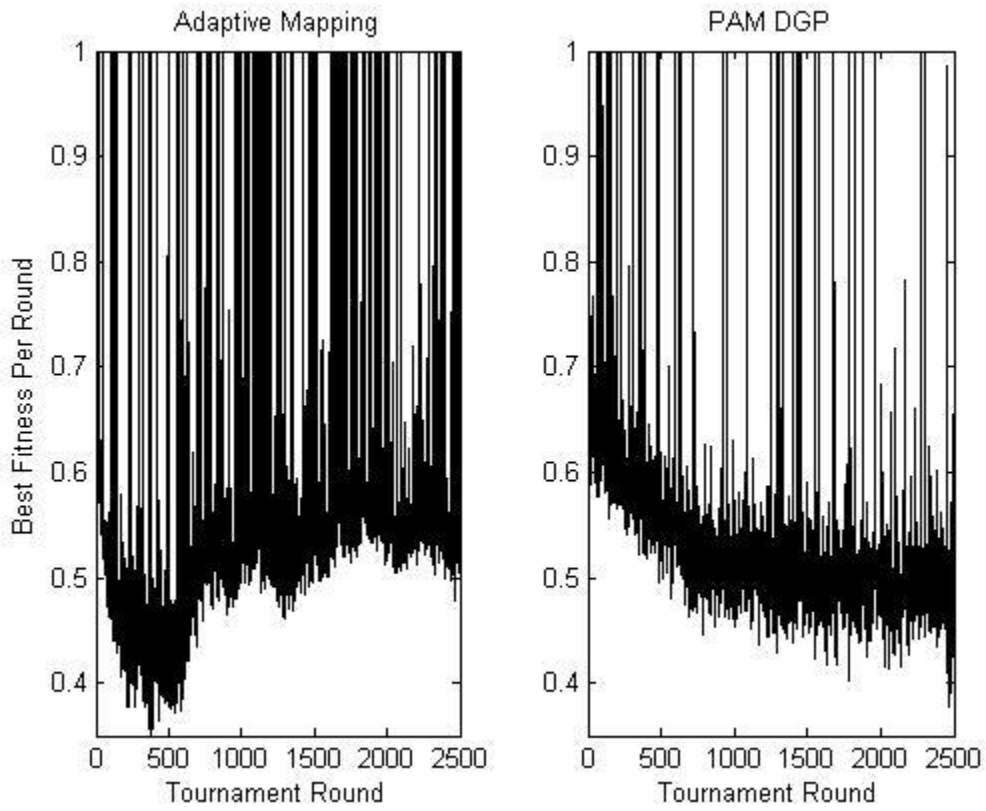


Figure 5.8. Mean best fitness per round over 50 independent runs for PAM DGP (right) and the Standard Adaptive Mapping algorithm (left), population of 50, for the Hénon Mapping problem. Graph is restricted to the fitness interval [0.35, 1.0] for clarity.

The mean final fitness for Standard Adaptive Mapping, PAM DGP, and Traditional GP is given below in Figure 5.9. No algorithms produce statistically different average mean squared errors over 50 independent trials given a 0.95 confidence interval. PAM DGP and Standard Adaptive Mapping perform on par with Traditional GP for this problem given the metric of mean final fitness: there is no statistical difference at the 0.95 confidence interval (p-value of 0.891 for PAM DGP and Traditional, 0.933 for PAM DGP and Standard).

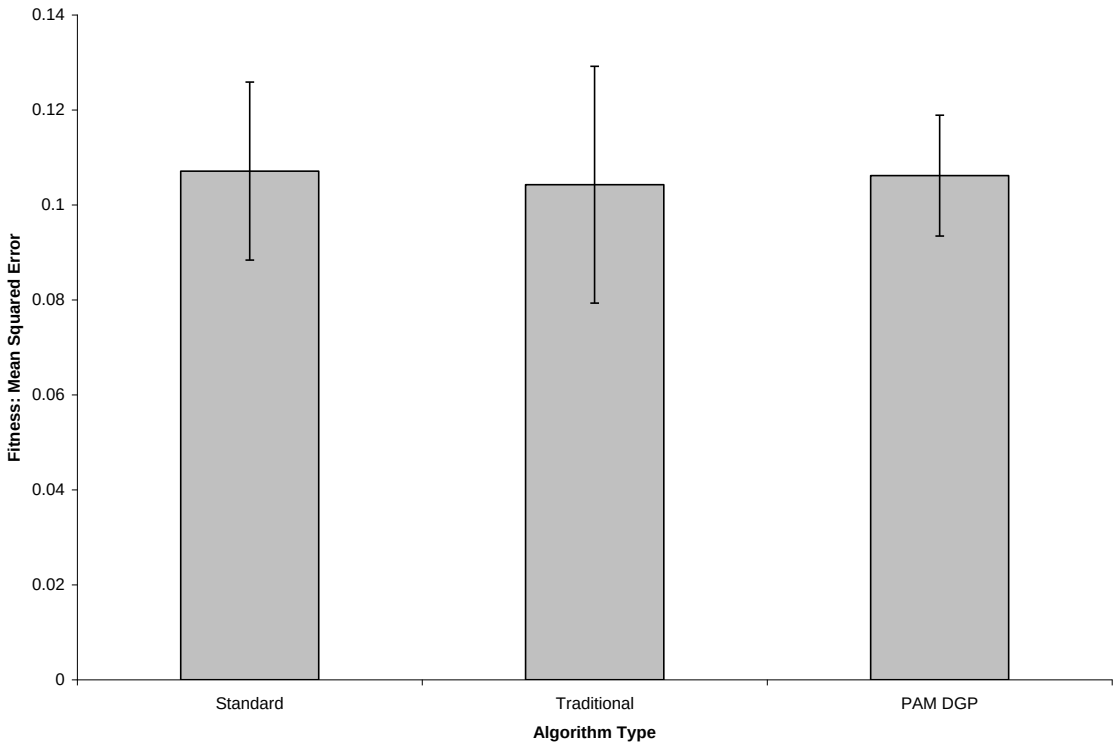


Figure 5.9. Mean best fitness achieved after maximum rounds for PAM DGP, Standard Adaptive Mapping DGP, and the Traditional GP algorithm over 50 trials for the Hénon problem. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval.

Given the fitness/error rate of PAM DGP on this problem, it is worthwhile to determine whether the algorithm is able to generate a reasonable approximation to the

Hénon mapping. The program of the individual produced by the PAM DGP trial with lowest error rate (mean squared error 0.0650), with introns (non-effective code) omitted, is shown in Figure 5.10 below. To contrast with the results of Margetts, he found a best mean squared error of 0.07 in 25 independent trials. Following the instruction interpretation in Section 5.2.2, the left argument for each function is one of four destination registers {*r0, r1, r2, r4*}, and the right argument is either an element of the fitness case {x_{t-1}, x_{t-2}} or a source register {*r0, r1, r2, r4*}.

```
SQRT r3 x_{t-1}
SQUARE r1 x_{t-1}
* r1 r3
CONST(0.8398565651941711) r0
- r0 r1
```

Figure 5.10. Best program produced by PAM DGP.

The program yields the uncomplicated mathematical function

$$0.8399 - \left[(x_{t-1})^2 \times \sqrt{x_{t-1}} \right] \tag{5.3}$$

The equation seems overly simplistic, for it uses only the last time step (x_{t-1}) of each fitness case in the calculation of x_t. (The original equation uses two time steps back for the calculation of the current x_t.) The plot of the solution in Figure 5.11 below, however, shows that it provides an accurate approximation to the Hénon mapping. The following chapter will present an alternative PAM DGP implementation that improves the accuracy further by providing an even closer fit to the actual function.

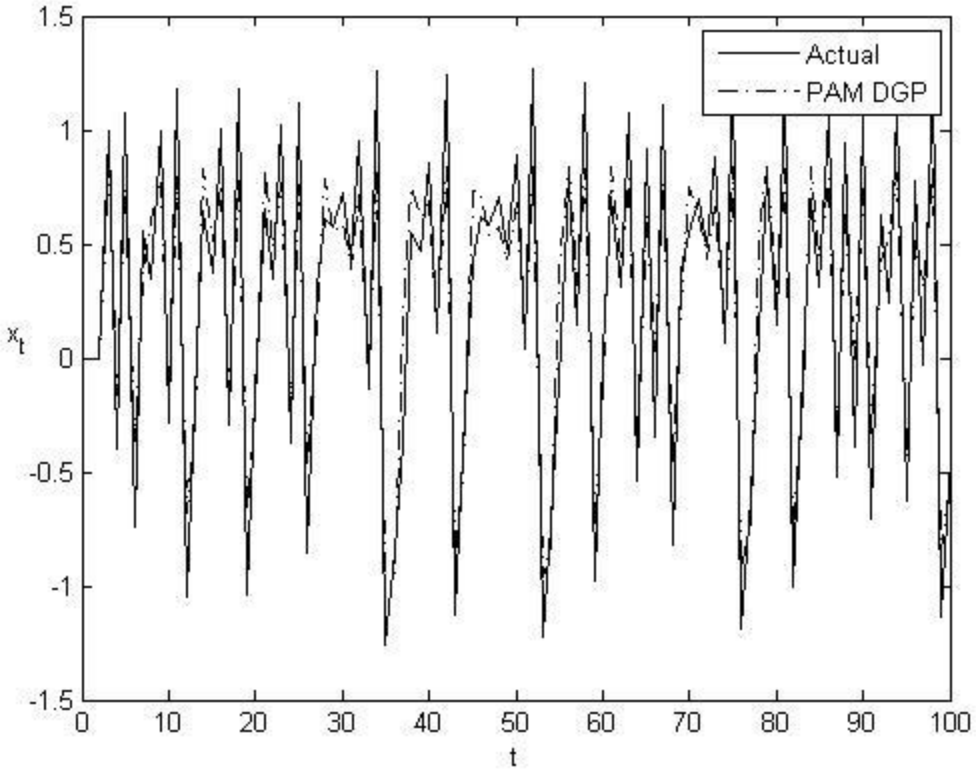


Figure 5.11. First 100 points (t, x_t) in the Hénon time series for the actual Hénon mapping and the best solution found by PAM DGP in 50 independent trials. The PAM DGP solution produced a mean squared error of 0.0650.

The operator content of the solutions as a percentage of total operators is shown in Figure 5.12 below along with p-values for the acceptance or rejection of the hypothesis that the symbol frequencies of the PAM DGP and Standard Adaptive Mapping algorithms are the same. There is no significant difference at the 0.95 confidence interval for all operators between PAM DGP and the Standard Adaptive Mapping. Overall, all algorithms chose a fairly even distribution of operator content to form solutions for the Hénon mapping problem. The actual Hénon mapping equation uses five out of the eight possible operators, so no pronounced emphasis on a small subset of operators was expected from these algorithms for this application.

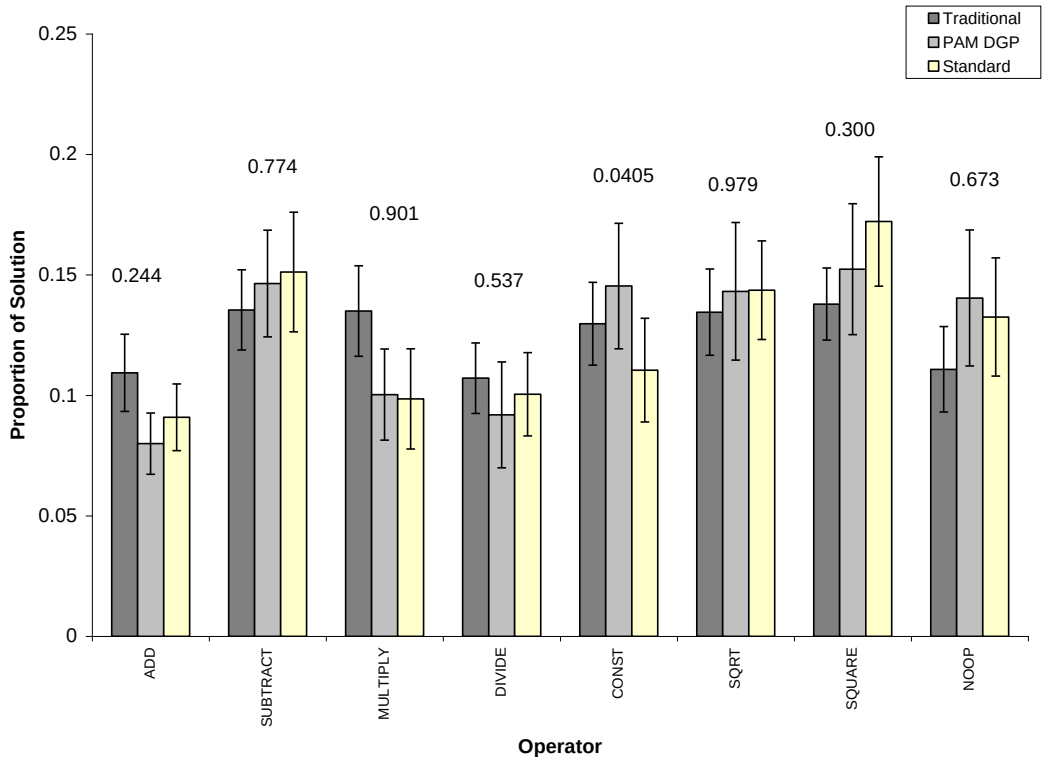


Figure 5.12. Mean operators as percentage of total solution over 50 trials for the Hénon Mapping Problem using PAM DGP, Standard Adaptive Mapping algorithm, and Traditional GP. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. P-values for PAM DGP and the Standard Adaptive Mapping are displayed above each set of data points.

5.3 Harder Regression Problem Summary

This chapter compared the performance of the Standard Adaptive Mapping DGP and PAM DGP on regression problems of increased difficulty compared to the MAX problem investigated in the last chapter. Upon application of PAM DGP to the Two Boxes problem, it was determined that performance of PAM DGP was not compromised by increasing population size from 8 to 50. At the higher population level, PAM DGP found more solutions to Two Boxes (2) than the Standard Adaptive Mapping DGP (0) or Traditional GP (1) following 50 trials of up to 150 000 rounds each. PAM DGP

outperformed the Standard Adaptive Mapping with respect to mean best fitness achieved over 50 independent trials, and performed competitively compared to Traditional GP considering the additional search overhead of PAM DGP. PAM DGP was also able to make better emphasis of the appropriate function set operators than Traditional GP, with comparable choices made to the Standard Adaptive Mapping DGP. For the challenging chaotic attractor Hénon problem, PAM DGP exhibited better long term performance of mean best fitness per tournament round over 50 trials than the Standard Adaptive Mapping DGP. However, no statistical difference was determined between any of the algorithms in terms of mean final fitness or with respect to the emphasis of operators in their solutions.

In terms of both harder regression problems in this Chapter, it is evident that PAM DGP continues to outperform the Standard Adaptive Mapping DGP on a number of metrics across three regression problems of increasing difficulty. The goal of Chapters 4 and 5 was to demonstrate that the PAM DGP algorithm empirically is able to outperform the Standard Adaptive Mapping DGP algorithm independent of genotype and mapping structures. We have clearly achieved this goal. We now move to further improve the PAM DGP algorithm by introducing a more developmental adaptive redundant mapping to take the place of Huffman-encoded mappings.

Chapter 6. An Investigation of an Adaptive Redundant Mapping Structure in the PAM DGP Framework

6.1 On Mapping Design: Redundancy and Neutrality

So far, the only structure used for mapping individuals in the PAM DGP framework has been the adaptive mapping scheme based on binary strings interpreted to yield a frequency for Huffman's compression algorithm. Keeping the mapping structure in PAM DGP and the Adaptive Mapping algorithm constant thus far has allowed a comparison of the algorithm component of PAM DGP with the algorithm component of the Adaptive Mapping DGP independent of differing mapping types. Having demonstrated the superiority of PAM DGP's algorithm component in the previous two chapters, we now move to improving the encoding process of its mapping individuals.

As mentioned in Chapter 2, Margetts's justification for the use of his Huffman-encoded mappings using the countingOnes function (Equation 2.1) is that small changes to the genotype will produce small changes in the phenotype [57, 59]. (In other words, the mapping is *phlegmatic*.) The usefulness of this property (for most problems) is expected to appear when, as genetic search progresses, the search closes in on an optimum. When this happens, there is less opportunity for the crossover operator to combine useful portions of individuals into a better solution. Thus, the mutation operator ought to be emphasized more in these later "exploitation" stages of search than the crossover operator to better implement local search. In addition, a small change in genotype causing a small change in phenotype (small change in fitness) ought to support the search around local optima. Given this line of reasoning, a small change in genotype

producing a large change in phenotype would discourage the necessary fine-tuned search at this stage.

There are a number of benefits that alternative mappings (encoding schemes) could provide that are not considered in works defining the Standard Adaptive Mapping DGP [57-59]. The most obvious of the benefits that are overlooked is that the Huffman encoding provides one and only one unique binary encoding to each member of the function set. For any given unique genome (binary permutation) in the genotype space, then, there is only one phenotype (binary permutation) in the phenotype space that corresponds to it. That is, there is a one-to-one mapping from genotype to phenotype and the mapping is not *redundant* at all. As mentioned in Chapter 2, the genetic code in nature is redundant (or, as biologists say, it is "degenerate"). Practical algorithm engineering considerations aside, then, redundant mappings are more developmentally accurate. But we will see that these mappings will prove more empirically beneficial when incorporated in PAM DGP.

Given the description of a redundant mapping provided by Keller and Banzhaf in [45], we consider a mapping (encoding) to be *redundant* if it can map more than one codon (genotype binary sequence) onto the same symbol. We should be careful to be clear here; we are speaking of redundancy at the level of the encoding. That is, a mapping/encoding is redundant when it allows more than one binary sequence to map to a single symbol. However, if any member of the function set is mapped to by more than one genotype, then it is the case that more than one distinct genotype (entire binary sequence representing an individual) can map to the same phenotype. (All that is necessary for this to occur is that the two genotypes be identical except for a single binary

subsequence where the differing subsequences corresponding to the same function.) Thus, if a representation is redundant at the mapping encoding level, then it is also redundant at the level of the genotype-phenotype individuals themselves. There is thus little concern about confusion, for the literature typically calls a genotype-phenotype mapping *redundant* if it maps multiple genotypes to the same phenotype [5, 25, 26, 45-47, 84-87].

The genotype-phenotype mapping is highly redundant in nature: many different genotypes result in phenotypes of comparable functionality [5]. According to the neutrality theory of evolution [48], most of natural evolution at the molecular level is due to mutations that are practically neutral with respect to selection. That is, variations in genetic material are typically neither advantageous nor disadvantageous, allowing evolutionary exploration of alternative genomes without a severe fitness cost to individuals possessing the alternate genomes. Examples of redundancy in nature include 64 codons of our genetic code mapping to only 20 amino acids, and the interaction of molecules to form an organism [87]. The artificial analogue of this neutrality allows the escape of local optima in search space [25, 26, 84-87]. This is accomplished by the neutrality allowing transitions between phenotypes that would not have been possible without being neutral with respect to the current phenotype. (That is, the mutation would not have been able to survive via the individual due to the likely lower fitness it would have caused the individual had the phenotype not remained unchanged following the mutation.) Given changes elsewhere in the genotype, however, a new and potentially fitter phenotype can now be reached given the presence of the neutral mutation. In the search space, these neutral mutations produce sets of genotypes that map to the same

phenotype and are connected via single point mutations. By moving throughout these neutral networks that permeate the search space, a population can find an area of the genotype space that allows increases in fitness and entrapment in local optima is prevented. Banzhaf's original work [5] proposing the redundant genetic code-based GPM established that genotypic diversity across high fitness individuals is afforded by the GPM's redundant representation using a measure of Hamming distance between all genotypes to the best genotype. Shipman, Shackleton, Ebner and colleagues [25, 26, 84-87] have also performed several investigations on quantifying redundancy and determining the correct types of redundancy to produce, and facilitate traversal of, neutral networks to escape local optima.

By introducing a redundant encoding into the PAM DGP framework, we can expect the added benefit of neutral mutations. However, the main motivation behind the introduction of redundant mappings is to facilitate search by better emphasizing particular members of the function set over others by actually reducing the number of functions in the set that are considered. That is, we look at the direct benefit of redundancy at the level of the genetic code. Indeed, redundancy in the genetic code itself is believed to reduce the exponentially scaling search for an optimal genome (related to fitness) to a polynomial scaling search [42]. While potentially including the benefits of increased connectivity in search space and more efficient search via neutral mutations, the investigation of this work aims to show that redundant mappings in DGP directly cause the search to become more efficient by eliminating unnecessary or detrimental functions as the function set is adapted for the problem. In terms of a developmental analogue, the investigation simply focuses on demonstrating that better genetic codes are

produced by only permitting particular amino acids to be represented collectively by codons rather than allotting an amino acid for each of 64 available codons (instead there are only 20 amino acids for the mapping in nature). The analysis on the benefit of the redundancy in mappings has used fixed mappings [25, 26, 84-87], with the exception of Keller and Banzhaf's DGP implementation with individual pairings of mapping and genotype [45, 46]. We now introduce, for the first time, an adaptive redundant mapping for use in a coevolutionary system of separate populations.

6.2 Introducing an Alternate Mapping Choice for PAM DGP: The Adaptive Redundant Mapping

To review, the only scheme used for mapping individuals in the PAM DGP framework has been the adaptive mapping scheme that uses binary strings interpreted with the countingOnes function (Equation 3.1), which simply sums the ones in a binary string, to yield a frequency. The frequencies correspond to each member of the function set, and the Huffman encoding scheme determines the encodings of each symbol of the function set using those frequencies. The Huffman encoding, based on the frequencies dictated by the countingOnes function, thus determines the emphasis placed on each member of the function set. The complete structure and process was described in Section 2.3.1, with the interpretation of Huffman-encoded individuals reviewed conceptually in Figure 6.1 below.

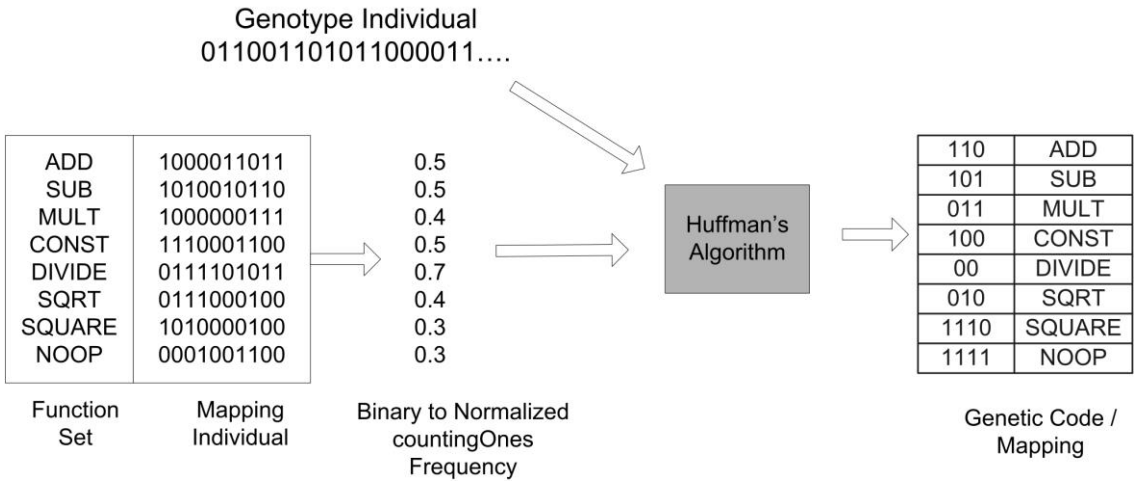


Figure 6.1. Huffman mapping encoding process.

In contrast to Huffman encoding, the mapping scheme of Keller and Banzhaf [5, 45-47] provides a more transparent, but severe, method of emphasizing members of the function set. As described in Section 2.2.1, each member of the function set may be encoded with more than one bit sequence. In other words, any given member of the function set can be *redundantly* encoded. The emphasis of a member of the function set then results from the number of bit sequences corresponding to it, as well as the content of those bit sequences (the more times the sequence is occurring in the building blocks of the fittest individuals, the better). It is possible for a mapping to dictate that certain function set symbols are never to be read when interpreting the genotype, or even in the extreme case that only one function is ever to be read. In contrast, the adaptive mapping scheme using Huffman encoding always allows some possibility that the symbol which ought to be read the least (has the lowest frequency in the mapping) still gets interpreted from a genotype individual's code. That is, Huffman encoding ensures that every member of the function set gets some unique encoding assigned to it, although members

of the function set with low frequencies in the mapping will be given more obscure, often longer, encodings.

Keller and Banzhaf introduced the concept of redundant mappings where each genotype individual is paired with its mapping for the entire tournament [45-47, 57]. This scheme is not compatible with either the coevolutionary architecture of the Adaptive Mapping algorithm or the PAM DGP framework. Thus, we design an *adaptive redundant mapping* for the PAM DGP framework with an associated encoding mechanism and structure suited to the PAM DGP algorithm. The proposed adaptive redundant mapping is summarized by Figure 6.2. Firstly, a list of binary sequences is chosen so that each symbol of the function set has the potential to be represented by one or more unique fixed length bit sequences. That is, a mapping individual is set to consist of $b \geq s$ binary strings of length 10, where b is the number of binary sequences required to represent a function set of size s. For example, a function set of size 4 would require a list of four sequences of 2 bits each, and a function set of size 7 would require a list of eight sequences of 3 bits each (with one extra encoding). Instead of each of the binary strings in the mapping individual representing frequencies, as in the Huffman encoding, they directly represent indices of the ordered function set. Each of the b 10-digit binary strings is interpreted by converting them to their decimal representation and normalizing to the range [0…1]. They are then mapped onto an ordered function set *index* (where indices are numbered 0 to $s - 1$ for a function set of size s) by multiplying them by s and truncating to an integer value. (In the rare case where the normalized decimal for a binary string is 1, a reference beyond the last index is avoided by referencing the last

index.) The ordering of the function set is arbitrary upon initialization and remains unchanged throughout the algorithm.[5]

The countingOnes method is not used to interpret the mappings, because when mapped onto the ordered function set it would greatly favor members at the centre of the function ordering and very rarely choose members at the start or end of the ordered function set. (Consider that in a countingOnes interpretation of 10 bits, there are many possibilities of encoding a frequency of 0.5 where five of the ten bits are ones, but only one possibility of encoding the frequency of 1.0 where all bits are set to one.) Using normalized binary to decimal conversion, however, there is a uniform chance of choosing any value from 0.0 to 1.0.

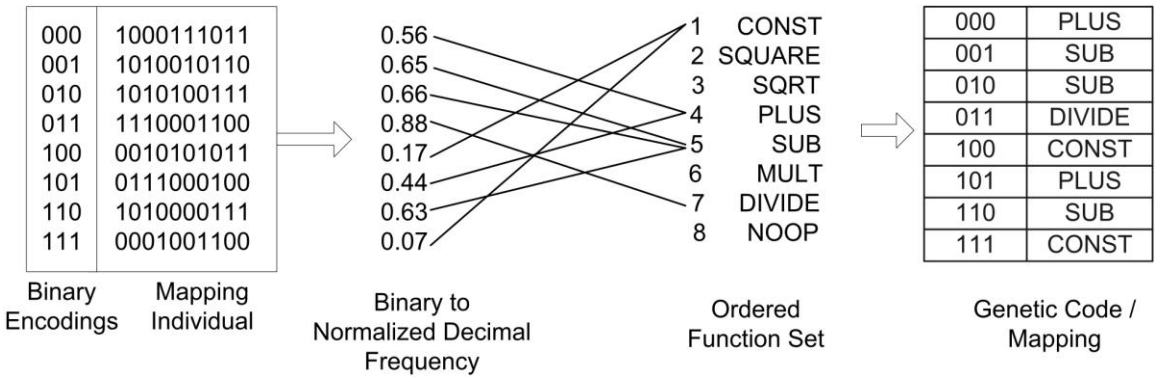


Figure 6.2. Adaptive redundant mapping encoding in PAM DGP.

[5] Note that this approach is a *direct* binary encoded mapping, and is not at all similar to Bean's Random Keys GA [9] that uses a random search space as an intermediary between genotype and phenotype.

6.3 Properties and Benefits of the Adaptive Redundant Mapping

An elaboration on the term *redundancy* as it applies to the adaptive redundant mapping may be helpful to avoid confusion. As mentioned previously, we adopt the term *redundant* in the sense of Keller and Banzhaf [45] in particular: a mapping is redundant if it "*may* map more than one codon onto the same symbol." We call our adaptive mapping redundant because it has the *potential* to map more than one genotype to a single phenotype; although it may not do so in all cases. By allocating distinct binary encodings to each symbol of the function set when the problem's function set can be represented by exactly s binary sequences ($s = b$), there is no redundancy introduced at all. If the problem's function set cannot be represented by exactly s binary sequences (it is required that $b > s$), then there will be some problem-dependent minimal degree of redundancy. The maximum redundancy for every function set of size s (every problem) is always the case where every function set symbol has the same encoding (although this would not allow a very fit solution in most cases).

Because the mapping is adaptive, it can range from the problem-dependent lower limit of redundancy to the upper limit throughout the algorithm. Thus, the adaptive redundant mapping can trim the function set when necessary or keep all function set symbols. Moreover, the adaptive redundant mapping has the added benefit that it can adaptively set its level of redundancy and may for some problem cases produce solutions that opt not to use redundancy at all—recalling the definition adopted from Keller and Banzhaf [45], it may (or may not) map more than one encoding onto the same symbol. Mathematically speaking, the mapping is non-injective and non-surjective: every element

of the function set can be mapped to by one or more binary sequences, but not every element of the function set will necessarily have a binary sequence mapped to it, respectively.

A justification of the nature of the encodings and the structure of mapping individuals presented in this section may be in order. The encodings described use strings of ten binary digits that map *directly* to an index of an ordered function set. The representation of digits could be shortened so that only the required number to enumerate the indexed function set is used. However, for the purpose of easily adapting new problems of various function set sizes to the algorithm, the binary representation of reals in the interval [0, 1] is used. Another reason behind this choice of 10 binary digits to specify each index is that it increases the chance of neutral mutations in the mapping space: point mutations have less of a chance of causing a change of index with respect to each binary encoding. That is, a point mutation in a space of ten binary digits (if in the lower portion/rightmost of the string) will not likely change the index to which the binary digits correspond. However, if there are only enough binary digits at each location to cover the binary representations of the indices of the function set, each point mutation will definitely cause a change in context. To put it another way, the representation of the redundant mappings allows for higher redundancy between the mapping "genotype" and the mapping "phenotype" (mapping phenotype actually being the genotype-phenotype mapping for the *overall problem*). There are actually two levels of redundancy/neutrality in the PAM DGP algorithm that can be considered: there is a level between the genotype population and phenotype for the overall problem, and for the mapping genotype and the genetic code it represents (an intermediary phenotype, if you will). The effect of using

this higher redundancy in the problem representation (letting genotypes map to more than one phenotype) is that there will be the opportunity for neutral mutations to escape local optima.

6.4 Computational Complexity Considerations

Since the Redundant mapping directly maps to indices of the ordered function set, and Huffman's algorithm need not be employed to evaluate the content of the mappings, there are savings in terms of computational complexity in addition to the anticipated benefit of the new mapping reducing the function set size in difficult problems. Figure 6.3 below shows the relevant portions of the PAM DGP algorithm and the computational complexity now that Redundant mappings are used. The computational expense difference between PAM DGP using Redundant mappings shown below in Figure 6.3, and PAM DGP using Huffman mappings shown in Figure 3.11, is essentially that there is no call to Huffman's algorithm. Now that a direct mapping into the function set is used, there is a computation saving of $O(n + n \log n)$.

The overall computational complexity can be broken down as follows: There are n tournament rounds of n genotype-mapping pairings. Each pairing will be evaluated in linear $O(n)$ time by executing the genotype's program to determine fitness based on the genotype program. Translation of the mapping components for each of the n pairings must take place before the genotype program can be interpreted (even if the mapping individuals are not explicitly evaluated for fitness). The determination of each of the n directly mapped encodings is done in $O(n)$ time. Combined, each pairing takes $O(n + n) \approx O(n)$ time. There is a sort of the n pairings in each tournament round, giving a

complexity of $O(n \log n)$ for heap sort. Breeding of the pairings is done in linear $O(n)$ time. The table of probabilities does n updates for each of n individuals, done in $O(n^2)$ time. The overall complexity of PAM DGP using Redundant encodings is thus $O(n\,(n + n + n \log n + n + n^2))$, or

$$O(\, n^3 + n^2 \log n + n^2) \in O(n^3), \tag{6.1}$$

which is a savings of $O(\log n)$ time compared to PAM DGP using Huffman-based mappings.

```
For n rounds                              O(n)
      Execute mapping encodings             O(n)
      Evaluate geno-map pairs               O(n)
      Rank n geno-map pairs                 O(n log n)
      Breed n geno-map pairs                O(n)
      Update table                          O(n^2)

Overall complexity:
                  O(n^3 + n^2 log n + n^2)  ∈ O(n^3)
```

Figure 6.3. Derivation of computational complexity for PAM DGP using Redundant mappings.

6.5 Comparative Mapping Performance for Previous Regression Problems

6.5.1 MAX Problem

The remainder of this chapter compares Huffman encoded mappings and Adaptive Redundant mappings just introduced in the PAM DGP framework for all regression problems previously examined: MAX, Two Boxes, and the Hénon map. All algorithm parameters remain identical to those described in previous chapters

(particularly in Tables 4.1, 5.1, and 5.3); only the mapping structures are being compared. Traditional GP is shown in all results as a benchmark against which to gauge performance. Figures 6.4-6.8 show the mean best fitness over 50 independent trials per tournament round for the MAX problem for a population of 8. Since infinity cannot be plotted, the fitness for any trial not yet achieving success at a given round is used to determine the mean. Figure 6.9 indicates the efficiency with which Traditional GP and the two mappings in PAM DGP located solutions in terms of mean tournament rounds.

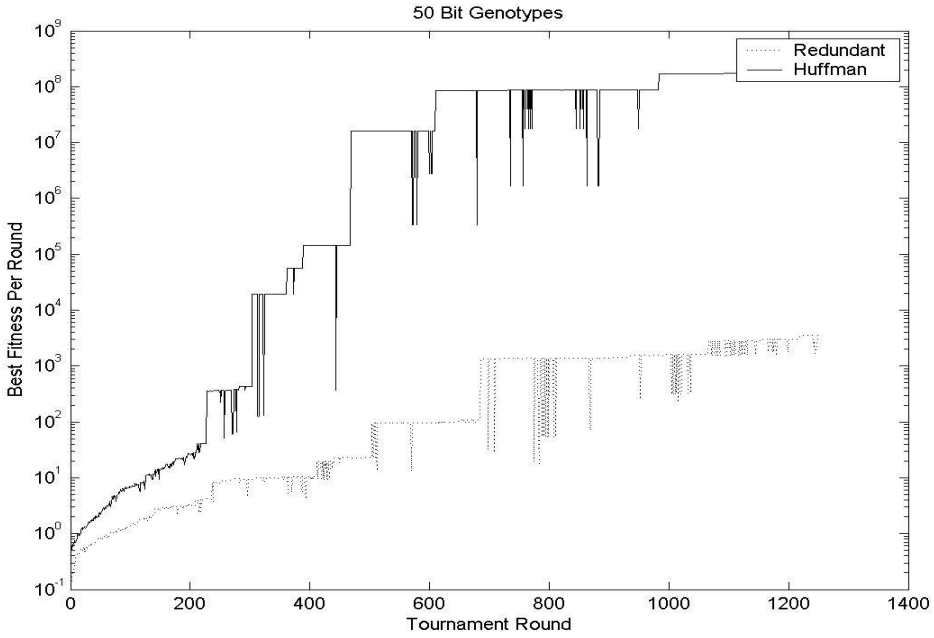


Figure 6.4. Mean best fitness per round over 50 independent runs for PAM DGP with Huffman and Redundant mappings, population of 8, on the MAX problem for 50 bit individuals.

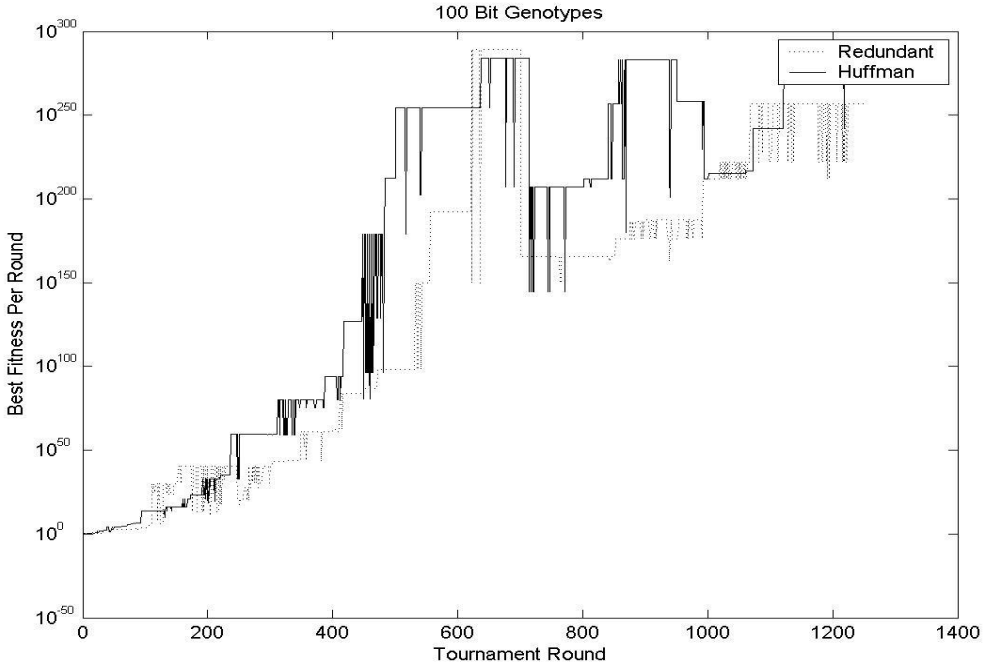


Figure 6.5. Mean best fitness per round over 50 independent runs for PAM DGP with Huffman and Redundant mappings, population of 8, on the MAX problem for 100 bit individuals.

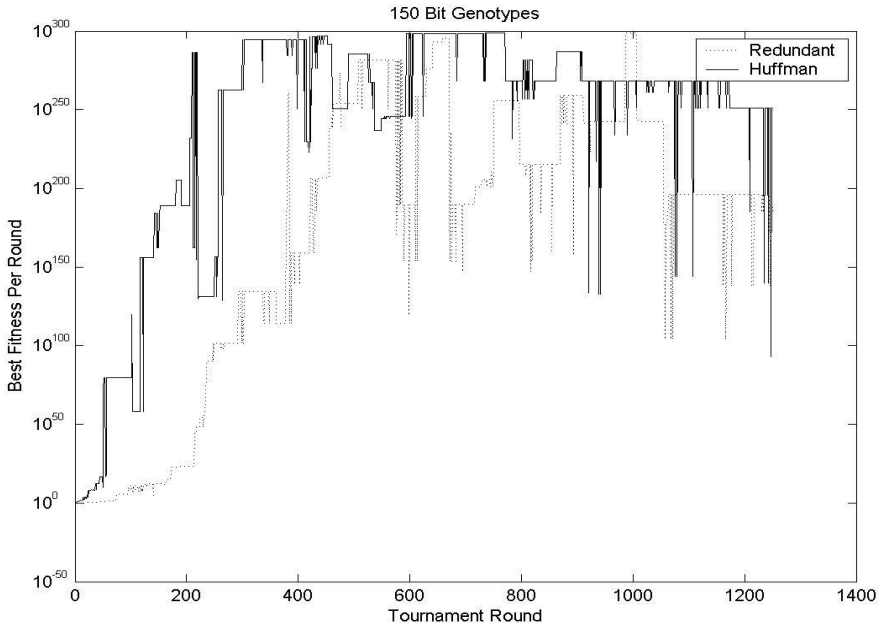


Figure 6.6. Mean best fitness per round over 50 independent runs for PAM DGP with Huffman and Redundant mappings, population of 8, on the MAX problem for 150 bit individuals.

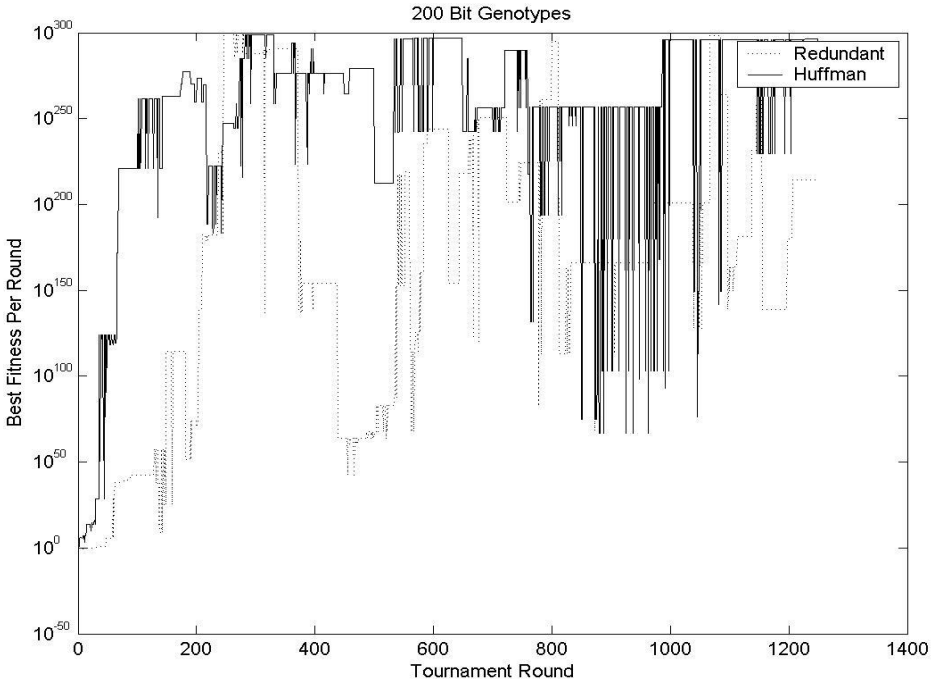


Figure 6.7. Mean best fitness per round over 50 independent runs for PAM DGP with Huffman and Redundant mappings, population of 8, on the MAX problem for 200 bit individuals.

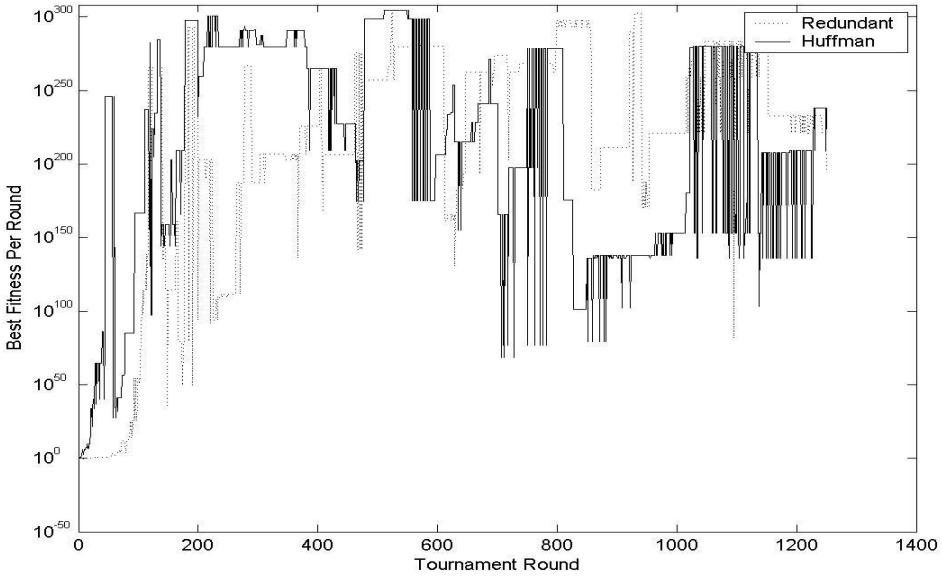


Figure 6.8. Mean best fitness per round over 50 independent runs for PAM DGP with Huffman and Redundant mappings, population of 8, on the MAX problem for 250 bit individuals.

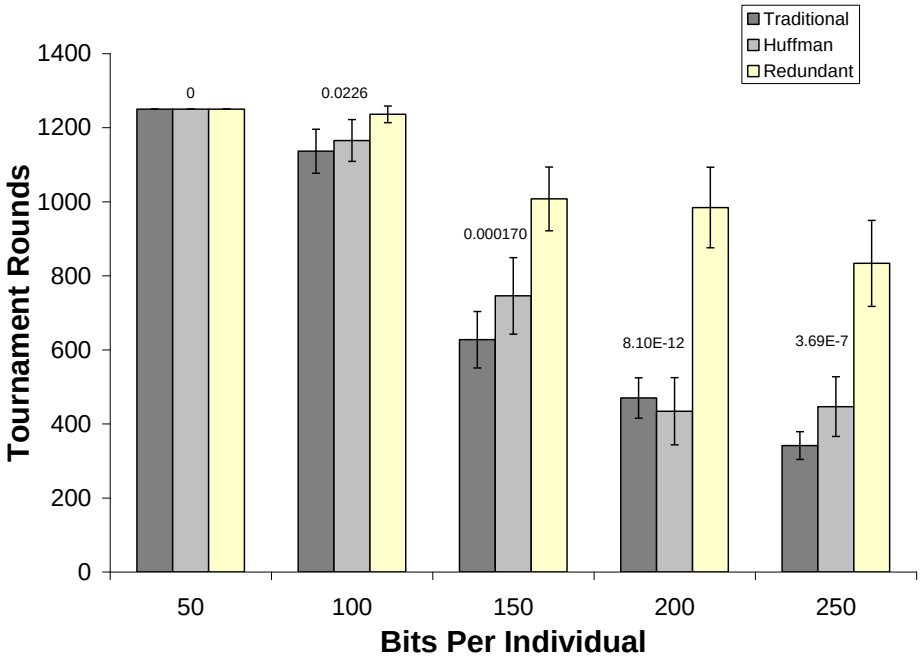


Figure 6.9. Mean number of tournament rounds (reaching maximum rounds or a solution) for Traditional GP, population 50, and Huffman and Redundant mappings in PAM DGP, population 8, over 50 trials for the Maximum Output problem. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. Corresponding p-values are displayed above each set of points for Huffman and Redundant mapping comparison.

Figures 6.4 to 6.8 show that the Huffman encoding (dotted line) achieves consistently higher fitness results across all tournament rounds, indicating that its more conservative symbol emphasis is more beneficial in the PAM DGP framework than the redundant mapping that emphasizes and eliminates function symbols throughout the search. Figure 6.9, using respective optimal populations for PAM DGP and Traditional GP, indicates that the Huffman encoding typically leads the PAM DGP framework to a solution faster than the redundant mapping, with that trend being more evident at higher bit levels where more solutions are readily found. P-values shown correspond to Redundant and Huffman mappings, indicating all results are significant at the 0.95

confidence interval. However, PAM DGP with either mapping typically does not outperform Traditional GP for this simple problem.

The number of total solutions found is in Figure 6.10 for the same trials. Figure 6.10 shows that the Huffman mapping produces more solutions than the redundant in the PAM DGP framework at all bit levels, but fails to outperform Traditional GP at any bit level. The results indicate that the Huffman mapping generates solutions more efficiently than the redundant mapping *in this very simple problem instance* by not eliminating function set members.

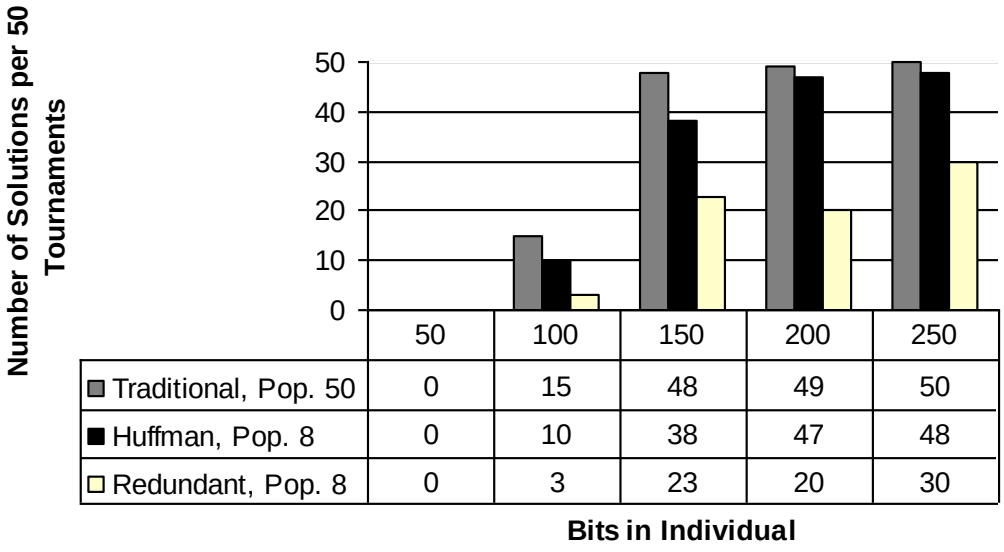


Figure 6.10. Number of Maximum Output solutions in 50 independent experiments, given Huffman-based and Redundant mappings in PAM DGP with a population of 8, and Traditional GP with a population of 50.

An analysis of function set operator content as a percentage of total operators in the solutions (with accompanying p-values) is below in Figure 6.11. Figure 6.11 shows that there is no significant difference in the operator content of the solutions for the two

mappings at the 0.95 confidence interval for 5 out of the 7 operators. Traditional GP has a similar distribution of operators to Huffman, with differences for PLUS and TIMES. Both Huffman and Redundant operators place a healthy emphasis on the DUP, PLUS, and TIMES functions—all very useful for creating large outputs. The redundant mapping uses more load constant to register (CONST) operators (significant at the 0.99 confidence interval), while the Huffman encoding uses more register to stack (R2S) transfers (also significant at the 0.99 confidence interval). Numerous CONST references are not necessary to solve the MAX problem, only an initial loading of a constant to be duplicated and multiplied is required. Further additional loading of constants are not necessarily detrimental, since the value for a loaded constant is placed at the top of the stack and could be used to increase the current output value by contributing to it when followed by the correct operators. The R2S operator favored by the Huffman mappings is not overly detrimental either; it simply pushes the item in the output register onto the stack where this number will tend to be large as a program of reasonable fitness is executed. In any case, the R2S operator is responsible for the third lowest proportion in the Huffman mapping solutions. Both mappings types in PAM DGP thus seem to make reasonable function set choices.

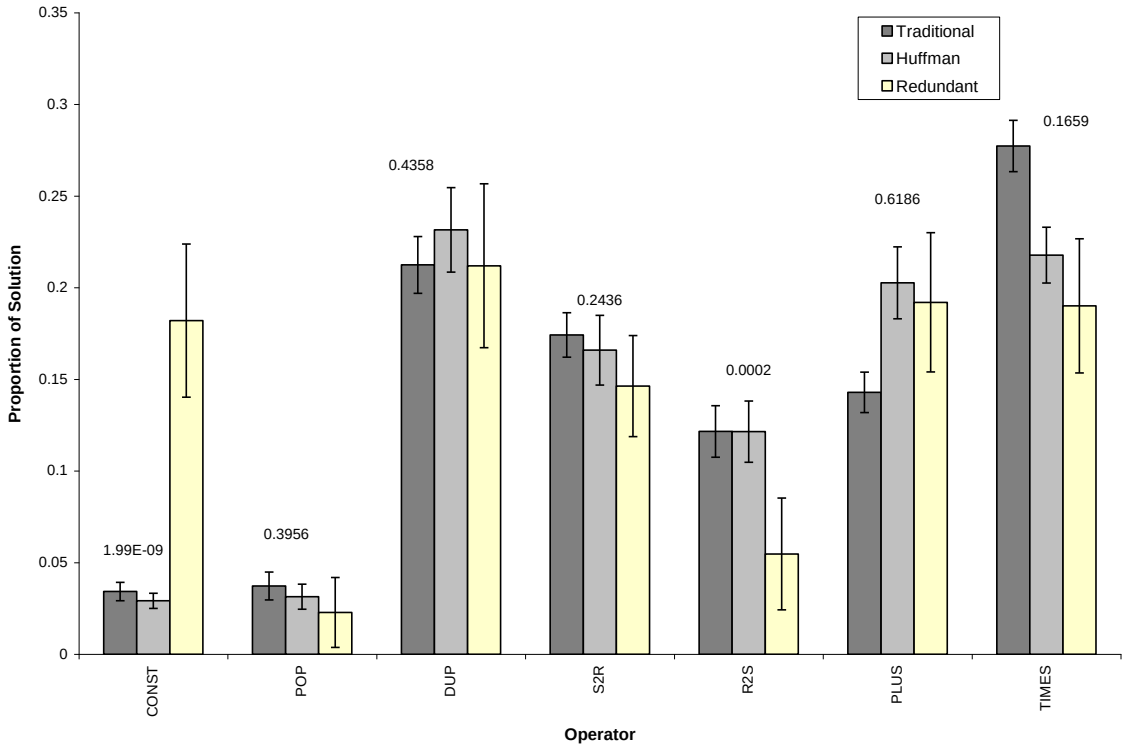


Figure 6.11. Mean operators as proportion of total solution over 50 trials for the MAX Problem using optimal populations for Traditional GP (population 50), and Huffman and Redundant mappings in PAM DGP (population 8). Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. Corresponding p-values are displayed above each pair of data points with respect to Huffman and Redundant mapping results for each operator.

Given the expected benefit of the redundant mapping trimming the size of the function set (in addition to appropriate symbol emphasis) to reduce search space, we now consider the number of unique operators used by each algorithm to form a solution. The boxplot is used to show the spread of data and the lowest operator cases produced to better indicate the function trimming tendency of the redundant mapping (simply plotting the mean and error would hide these outliers). Figure 6.12 below shows the boxplot for the number of unique operators used in a solution for Traditional GP, Huffman mapping PAM DGP, Redundant mapping PAM DGP, and the Standard Adaptive Mapping DGP of

Margetts and Jones. (The latter algorithm is included in this and similar boxplots in the following two sections, despite having already demonstrated that superiority of the PAM DGP algorithm, because this particular analysis was prompted by the introduction of the new mapping and hence has not yet been performed on the Adaptive Mapping DGP.) The Huffman mapping uses all 7 operators except for one solution containing 6. The Traditional GP uses 6-7 operators for all solutions, as does the Standard adaptive mapping algorithm. The redundant mapping typically uses only about 4-5 of the 7 operators, with one solution actually using 2 operators. Those 2 operators were actually the best two operators for the fastest generation of large numbers: R2S (register to stack) and DUP (duplication). The 2 operator solution was not discovered in time to generate the value of infinity within the 1250 round limit tournament, but it is significant that the most succinct and effective means of solving the Maximum Output problem was discovered using the redundant mappings. Overall, Traditional GP with static, globally applied mappings and Huffman mappings that used the bulk of the function set produced solutions more efficiently for this simple version of MAX problem than the Redundant mapping.

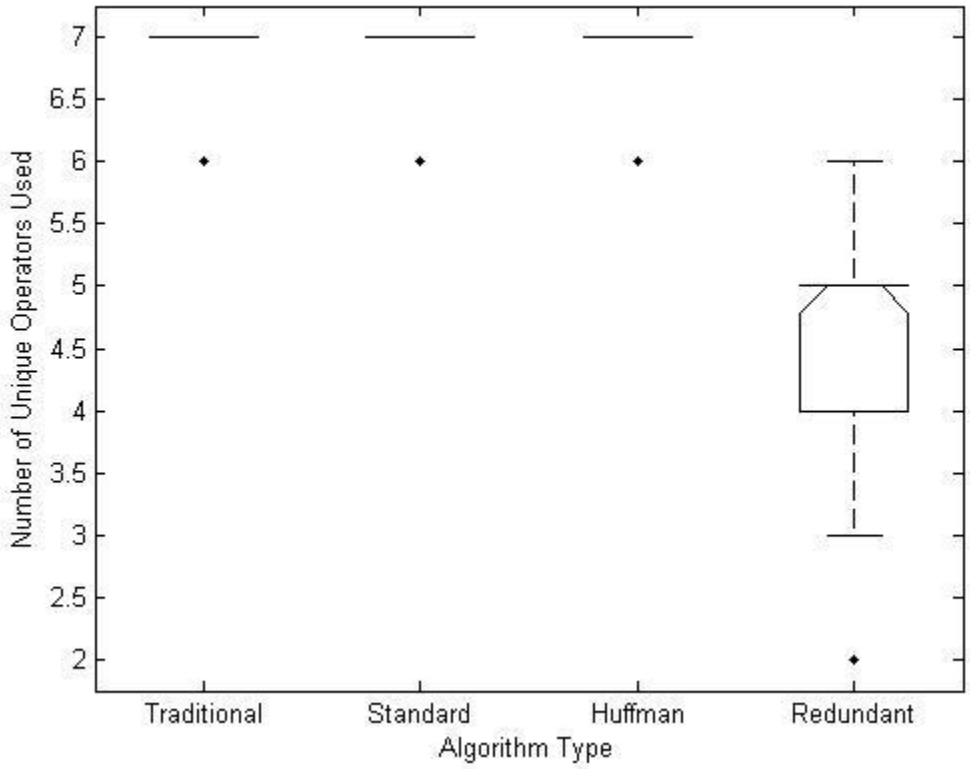


Figure 6.12. Boxplot indicating number of operators constituting each solution over 50 trials for the Maximum Output Problem using Traditional GP, Standard Adaptive Mappings, and Huffman and Redundant mappings in PAM DGP. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times interquartile range.

The Maximum Output problem, as construed in these experiments, is not a difficult problem for the adaptive mapping developmental systems to solve (Figures 4.7 and 6.10). The Huffman encoding seems to solve this simpler problem more quickly than Redundant encoding by using the entire (or near entire) mix of operators available. It is also noteworthy that none of the operators used in this problem are actually detrimental to accumulating large outputs—that is, there are no operators to decrement the accumulated value such as subtraction or division. There is thus less of a demand on the algorithm to

carefully select its operators; which is the real advantage behind the redundant mapping. It is also suspected that a lower number of solutions are found by the redundant mapping implementation due to the search time required in exploration of trimming the operator set, whereas the Huffman encoding only really needs to search for a solution while using all (or nearly all) of the available operators. A harder, more practical challenge is likely required to showcase the benefit of the adaptive redundant mapping where the search time spent in trimming the operator set is of benefit.

6.5.2 Two Boxes Problem

The harder regression benchmark, Two Boxes, is now attempted with redundant mappings. Parameterization of PAM DGP remains the same as outlined in Section 5.1.1, with the comparison of the two mappings in PAM DGP being done with a population of 50. The mean fitness of the best individual after the maximum number of rounds (150 000) or finding a solution over 50 independent trials is shown in Figure 6.13 below (lower fitness is better). As Figure 6.13 would indicate, no algorithm outperforms any other at the 0.95 confidence interval. However, Huffman encoding (as before, Section 5.1.3) and Redundant encoding both produce two solutions within the 50 trials of 150 000 rounds, whereas Traditional GP (as before) only produced one (Table 6.1).

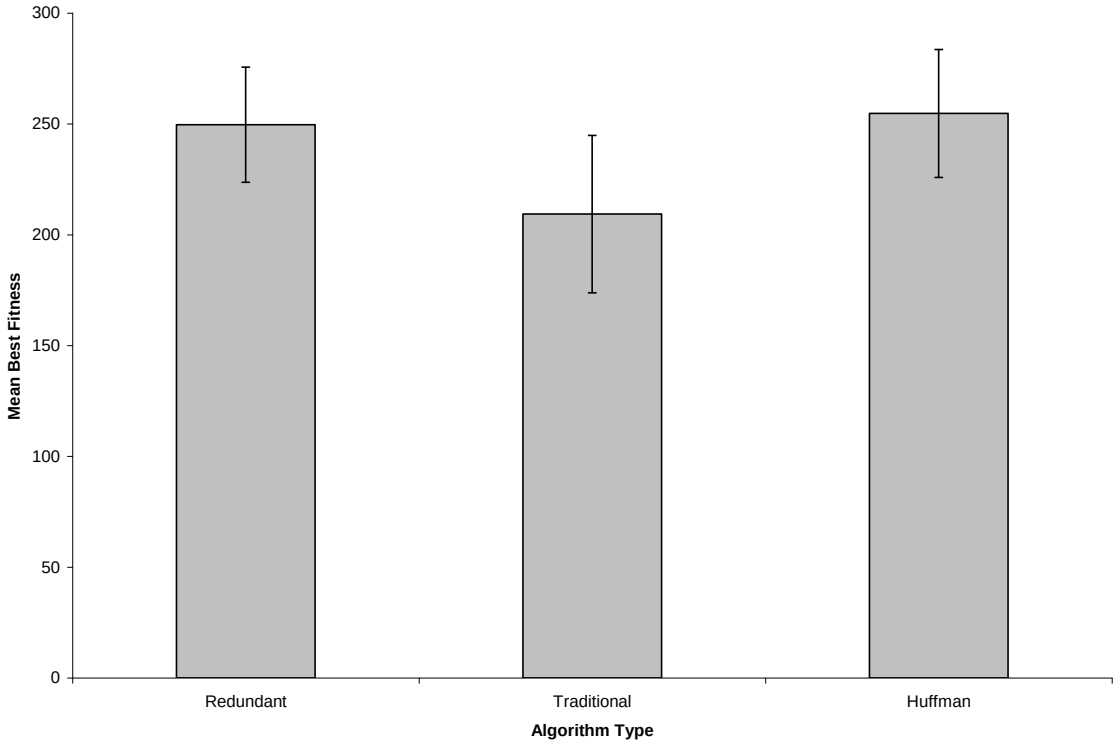


Figure 6.13. Mean best fitness achieved (after maximum rounds or a solution) for Traditional GP, Huffman mapping PAM DGP, and Redundant mapping PAM DGP over 50 trials for the Two Boxes problem. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. The p-value for the Redundant and Huffman mappings is 0.791.

Table 6.1. Two Boxes solutions for Traditional GP, Redundant mappings and Huffman mappings in PAM DGP.

Algorithm	Solutions per 50 Trials	Round Solution Found
Traditional GP	1	42004
Redundant PAM DGP	2	130892, 113046
Huffman PAM DGP	2	134743, 36161

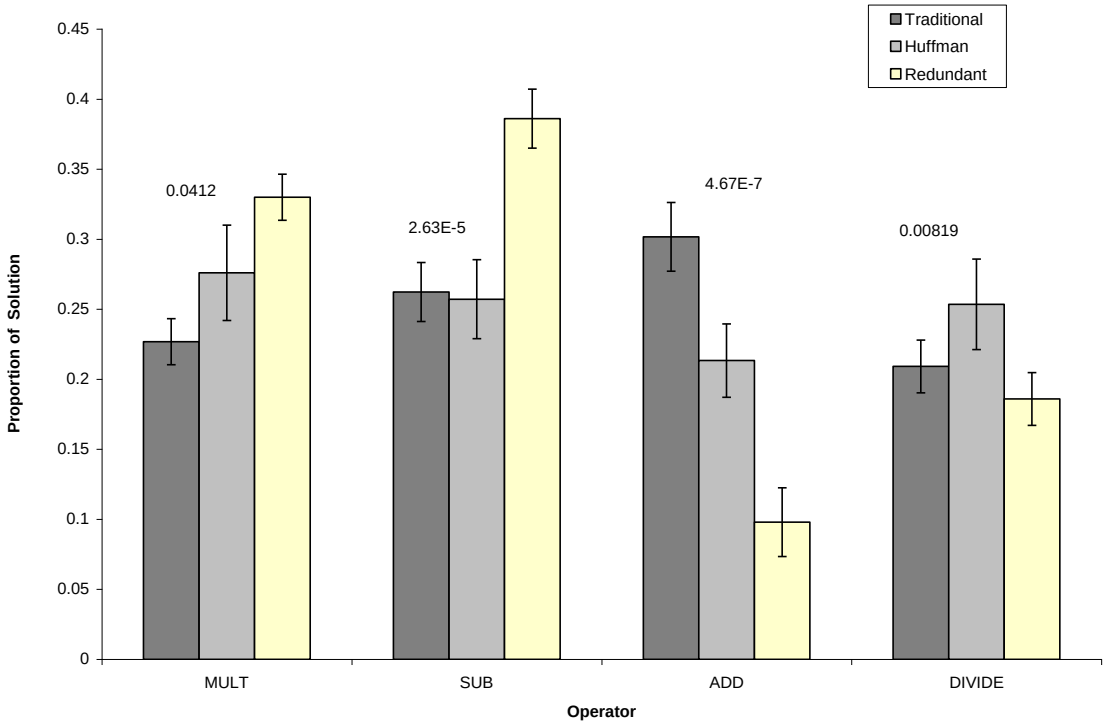


Figure 6.14. Mean operators as a percentage of total solutions over 50 trials for the Two Boxes Problem, population 50, using Traditional GP, Huffman mapping PAM DGP, and Redundant mapping PAM DGP. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. P-values corresponding to Huffman and Redundant mappings are displayed above each set of data points.

Operator content of the solutions as a percentage of total operators is shown above in Figure 6.14. There is a significant difference in the symbol content of solutions for the two mappings on the Two Boxes problem. All operator proportions for Huffman and Redundant are dissimilar at the 0.95 confidence interval, indicating that the mappings caused PAM DGP to choose different solution content. The redundant mappings successfully emphasize the only necessarily operators for the ideal function (multiplication and subtraction) to a greater degree than the other algorithms. Figure 6.15 below indicates that even with such a low number of available operators for the Two

Boxes problem (4 operators), the redundant mapping still managed to trim the median number of operators used in a solution to 3. In one case, the Redundant mapping PAM DGP used only the 2 correct operators (lower outlier in rightmost boxplot). The other algorithms used all 4 operators in all solutions.

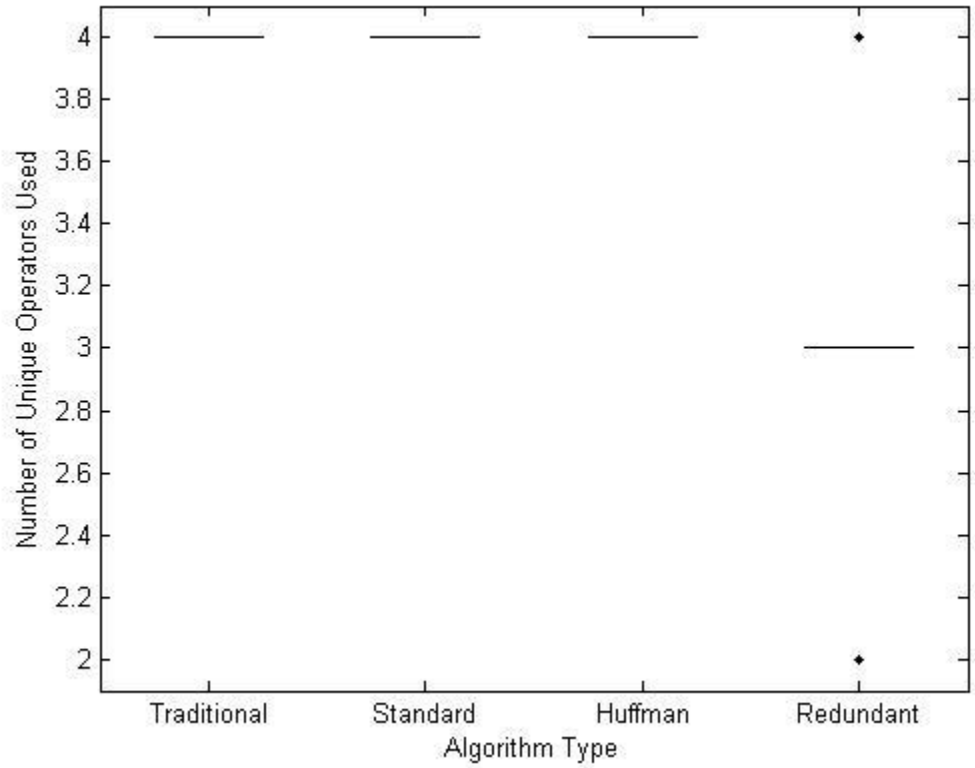


Figure 6.15. Boxplot indicating number of operators constituting each solution over 50 trials for the Two Boxes Problem using Traditional GP, Standard Adaptive Mappings, and Huffman and Redundant Mappings. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times interquartile range.

6.5.3 Hénon Mapping Problem

As expected based on previous results (last chapter), the Redundant mapping did not generate a solution for the Hénon mapping problem. However, this section will show that the redundant mapping did efficiently generate good approximations to the true function. The average best fitness per tournament round over 50 independent trials generated for both types of mappings (Huffmand and Redundant) is shown below in Figure 6.16. The Redundant mapping (right) begins with a better fitness (lower mean squared error) and maintains that lower fitness throughout the tournament and reaches lower error rates by later tournament rounds when compared with Huffman. This trend indicates that the Redundant mapping solution search is more efficient than its Huffman-based counterpart.

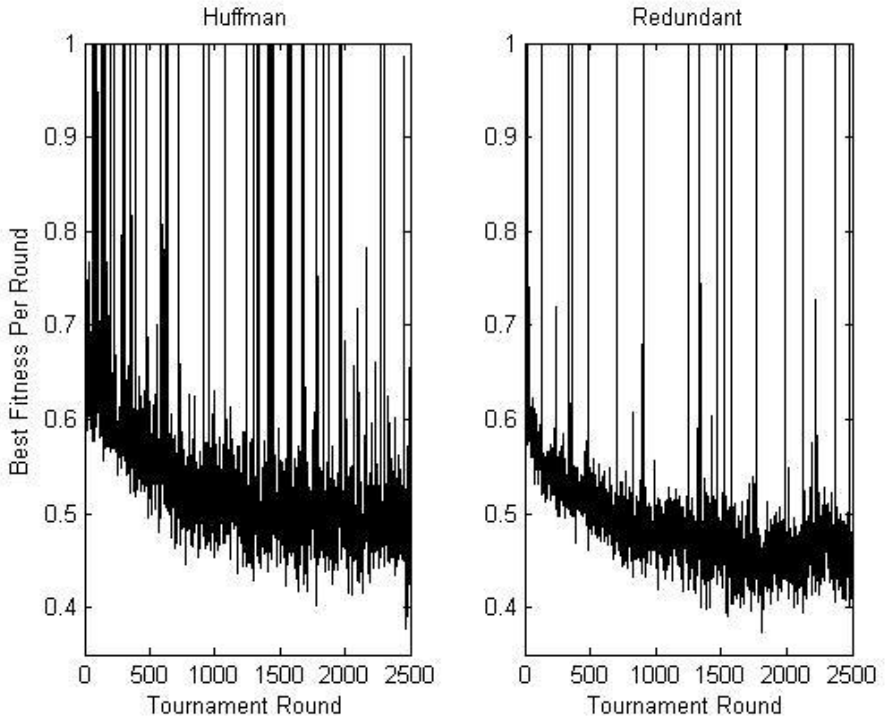


Figure 6.16. Mean best fitness per round over 50 independent runs for Redundant mappings (right) and Huffman mappings (left), population of 50, for the Hénon problem.

The best average fitness at tournament close over 50 independent trials is shown below in Figure 6.17. There is no statistically significant difference between the algorithms at the 0.95 confidence interval as evidenced by the plot. More precisely, the p-value for Huffman and Redundant is 0.111, between Traditional and Redundant it is 0.187. Considering the implications of both Figure 6.16 (above) and Figure 6.17 (below), despite the efficiency of the redundant mapping implementation during search, the final best fitnesses of the two mapping techniques (and fixed mapping of Traditional GP) discovered over the course of the tournament do not significantly differ. This is a reflection of the problem difficulty; the equation, after all, contains non-repeating real values and models a chaotic attractor whose orbit is not easily described [30].

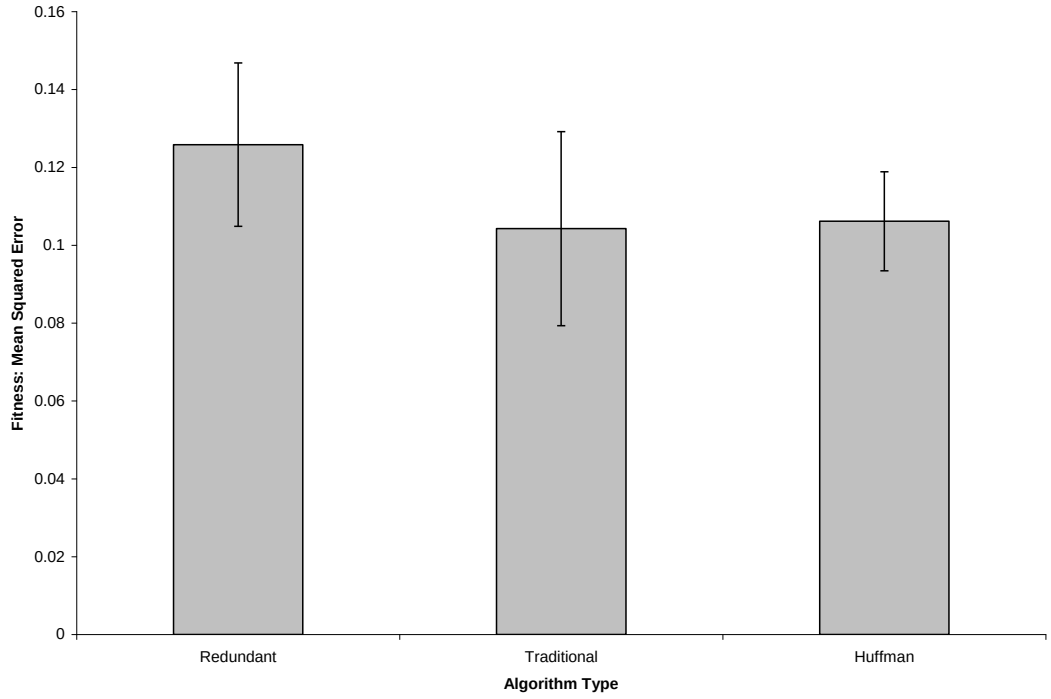


Figure 6.17. Mean best fitness achieved after maximum rounds for Redundant mappings, Huffman mappings, and Traditional GP over 50 trials for the Hénon Problem. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval.

While final fitness was not statistically different among algorithms, the fit of the best solution to the Hénon map produced by the Redundant encoding was found to improve over Huffman. The program corresponding to the best solution (effective code only) produced by redundant mappings in PAM DGP is given below in Figure 6.18. The left argument for each function is one of four destination registers {*r0, r1, r2, r4*}, right is either an element of the fitness case {x_{t-1}, x_{t-2}} or a source register {*r0, r1, r2, r4*}. The operation is performed using the value in the source register as the right operand, destination as the left. The result is placed in the destination register.

```
SQUARE r2 x_{t-2}
+ r2 x_{t-2}
SQRT r3 r2
SQRT r0 r3
SQUARE r3 x_{t-1}
- r0 r3
```

Figure 6.18. Best program produced by Redundant mapping PAM DGP.

The solution can also be expressed mathematically as

$$\sqrt[4]{x_{t-2}^{\,2} + x_{t-2}} - x_{t-1}^{\,2} \tag{6.2}$$

The mathematical equation reflecting the redundant mapping solution uses two previous time steps (x_{t-1} and x_{t-2}) to determine current value of x_t, in contrast to the Huffman solution (Equation 5.3) that only used the previous time step. In its use of the fitness case information, then, it is a more accurate reflection of the true Hénon mapping equation (Equation 5.2). This redundant encoding PAM DGP solution achieved a mean squared error of 0.0443, bettering the 0.0650 of Huffman encoding in the last chapter. The

redundant mapping also produces a better fit: the best redundant solution results in 164 hits where a hit is a raw error of ≥ 0.01 for a fitness case. In contrast, the best Huffman solution in terms of error that was examined in the last chapter only produced 87 hits. The redundant mapping solution is shown graphically to be a good approximation to the true formula in Figure 6.19.

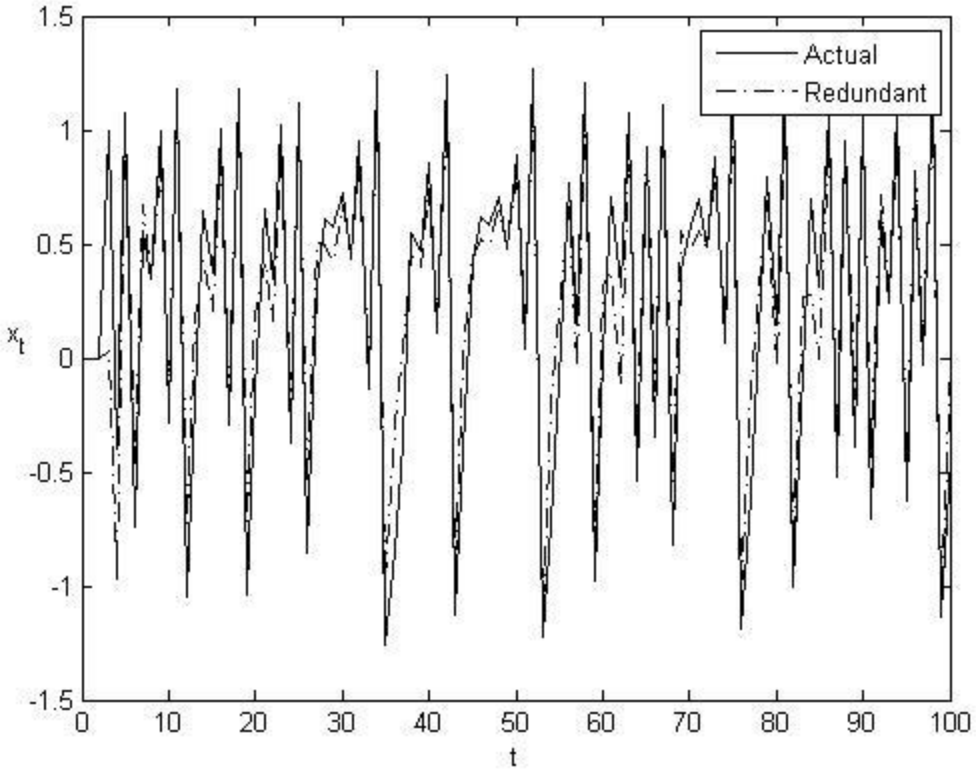


Figure 6.19. First 100 points (t, x_t) in the Hénon time series for the actual Hénon mapping and the best solution found by Redundant mapping PAM DGP in 50 independent trials. The PAM DGP solution produced a mean squared error of 0.0443.

Figure 6.20 indicates what subsets of operators are being emphasized by the redundant mapping trials. It is evident from Figure 6.20 that the redundant mapping does emphasize particular operators over others. No other algorithm significantly emphasizes particular operators (see also Figure 5.12 and associated discussion in Section 5.2.3). At

the 0.95 confidence interval, Redundant mappings result in the emphasis of subtraction and squaring compared to Huffman. It is also statistically significant (0.99 confidence) that the Redundant mapping puts less emphasis on multiplication and division compared to Huffman. All of the operators emphasized or used sparingly by the Redundant mapping PAM DGP feature in the actual equation, but observations regarding operator emphasis must be weighed with the fact that there are many ways of modeling the equation that do not necessarily require use of the operators in the true Hénon equation (consider Equations 5.3 and 6.2 for instance).

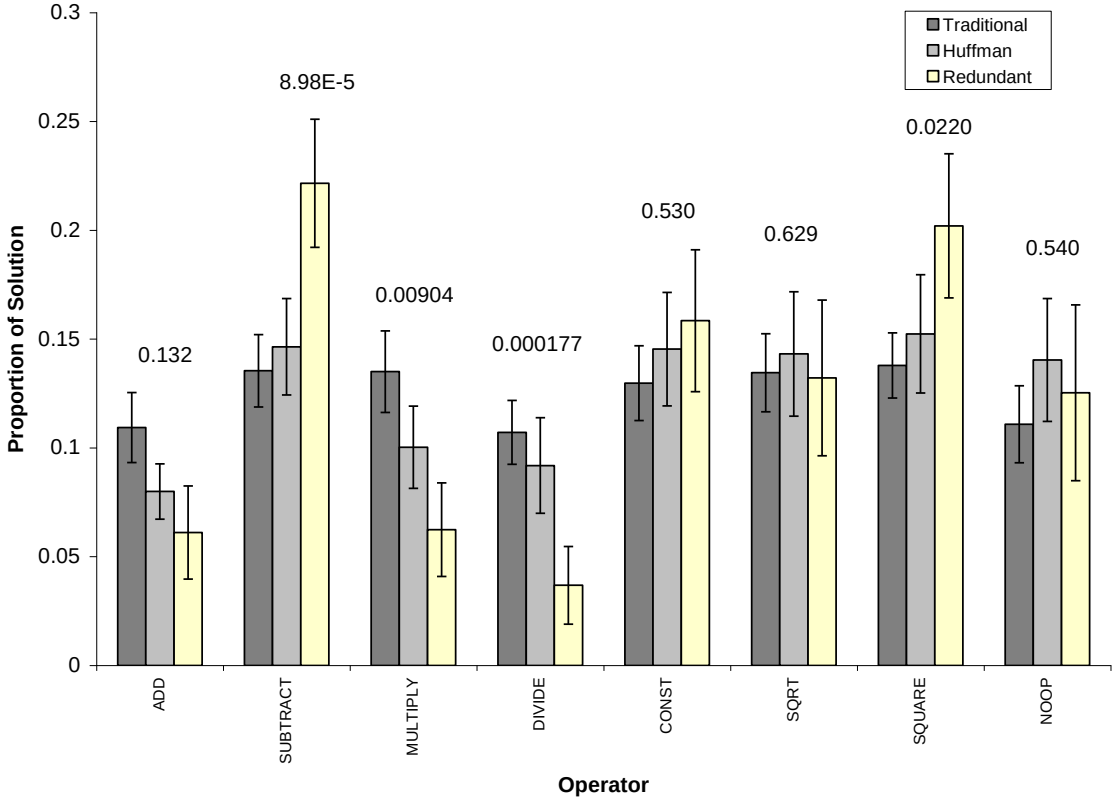


Figure 6.20. Mean operators as percentage of total solution over 50 trials for the Hénon Mapping Problem using Traditional GP, Huffman and Redundant mappings. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. Corresponding p-values are displayed with respect to Huffman and Redundant comparison.

Having completed the analysis of the operator composition of the solutions; the associated statistic of how many operator types are used in a solution completes the analysis of the effects of the redundant mapping on solution composition. Figure 6.21 shows the boxplot corresponding to the number of operators constituting a solution for all algorithms, verifying the trimming effect of the redundant mapping on the operator set. The redundant encoding does not produce a single solution that uses all 8 operators, using a maximum of 7 operators and as few as 4. In contrast, the Huffman encoding uses 8 operators in all cases except for three consecutively lower outliers. Traditional GP and Margetts's Standard Adaptive Mapping typically use 8 operators as well, with the Standard Adaptive Mapping having a smallest total of 6 operators used in a solution. Figure 6.21 shows clearly that the redundant mapping is producing good approximations by trimming the operator set included in solutions: the median for the Adaptive Redundant mapping algorithm (5 operators) actually corresponds to the lowest outlier of any other algorithm.

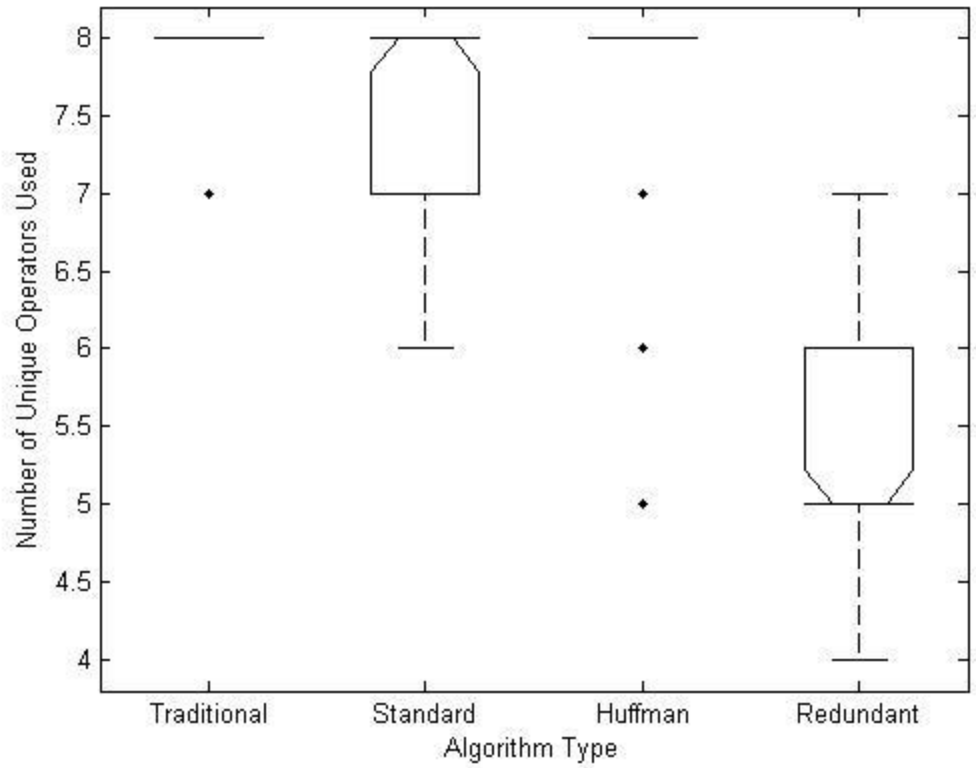


Figure 6.21. Boxplot indicating number of operators constituting each solution over 50 trials for the Hénon Mapping Problem for each algorithm. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times interquartile range.

6.6 Adaptive Redundant Mapping PAM DGP Summary

This chapter opened with motivations behind the introduction of an adaptive redundant operator to replace one-to-one (non-developmental) Huffman-encoded mappings in PAM DGP, followed by a description of the redundant mapping's structure and encoding procedure. PAM DGP with the new redundant mapping was then applied to all the regression problems of the previous chapters, namely the MAX problem, Two Boxes, and the Hénon map. The adaptive redundant mapping in PAM DGP did not outperform the other algorithms on the simple MAX problem. However, the MAX

problem actually involves a function set containing no operators that are directly detrimental to finding a solution, so any search using redundant mappings to trim the function set would be unnecessary and actually waste search time. This is not typical of most interesting problems, where the function set may contain symbols that ought to be emphasized or even eliminated from consideration. Analysis of the more difficult Two Boxes and the Hénon map regression benchmarks indicate that the adaptive redundant mapping performs better, and in no case is outperformed, by Huffman-encoded mappings on fitness-based metrics such as time (rounds) to solution, number of solutions (where applicable), and mean final fitness. Across all three regression problems, it is evident that PAM DGP using adaptive redundant mappings effectively trims function sets and emphasizes appropriate members of those trimmed sets to produce its solutions.

Chapter 7. Results using PAM DGP on Medical Classification Problems

7.1 Problem Definitions and Parameterizations

Having used regression benchmarks of increasing difficulty to establish the superiority of PAM DGP's underlying algorithm over the Standard Adaptive Mapping DGP (Chapters 4 and 5) and to investigate the benefits of an adaptive redundant mapping (Chapter 6), we now move to new problem domains in the present and the following chapter to showcase the abilities of PAM DGP using the adaptive redundant mapping. In this chapter, we examine two well known classification benchmarks using medical data sets from the UCI machine learning repository [66]. The first data set used contains Heart Disease data collected at the Cleveland Clinic Foundation [23]. The database contains 303 instances (164 negative, 139 positive), each consisting of 13 attributes, with a 14^{th} indicating positive or negative diagnosis. The second data set is 699 instances of Breast Cancer data (458 negative and 241 positive) obtained at the University of Wisconsin Hospitals, Madison [97]. Each instance contains 9 useful attributes, with a 10^{th} classifying it as positive or negative. The Heart and Breast data sets were selected from the repository because they were relatively small, have established performance levels (in [16] and others), and represent real world (rather than artificial) data.

Additional parsing of the file was performed prior to the experiments: Unknown values were replaced by the mean value of data recorded for the relevant attribute in the database, and a positive or negative classification was changed to '1' or '0' for all instances. Finally, the experiments used four-fold cross-validation to verify accuracy of the findings. The data set was partitioned so that 25% was used as a test set, with 75%

being used as the training set. Each of the four partitions used a unique set of instances for the training and test partitions, and both the training and test set retained a class distribution of the entire data set. For partitions 1-3 of the Breast data set, this resulted in 524 training instances and 175 test instances; for partition 4, there were 525 training instances and 174 test instances. For partitions 1-3 of the Heart data set, this resulted in 227 training instances and 76 test instances; for partition 4, there were 228 training instances and 75 test instances.

The division of the data and parameterization of PAM DGP are shown in Table 7.1 below. Mutation and crossover rates and types are kept the same as for the harder regression problems, with lower mutation and crossover rates for the mapping population than the genotype population providing a more stable background for solution search among genotypes. Preliminary experiments showed that no gain in training was accomplished for the Heart Disease data after 30 000 rounds, so this was chosen as the tournament limit for training the classifier. Breast Cancer was given a limit of 50 000 rounds for training. Note that the function set is larger than would typically be employed for a classification problem (for instance [16]), since we are interested in evaluating the ability of the mapping to suitably focus the search on the most applicable subset of instructions.

Table 7.1. Medical Classification Problems parameterization.

Tournament Style	Steady State, 4 individuals for each round
Training Rounds	Heart=30 000, Breast=50 000
Cross validation	4-fold, each with 75% Training and 25% Test
Data Set Characteristics	Heart: 303 instances, Breast: 699 instances
Function Set	+, *, -, % (protected), SIN, COS, EXP, LOG (base 10), SQRT, NATLOG
Genotype structure	Instruction sequence with 4 registers; 320 bits
Mapping structure	100 bits (10 bits per function set member)
Genotype mutation	XOR mutation, threshold = 0.5
Mapping mutation	Point mutation, threshold = 0.1
Genotype crossover	Equal-sized blocks, threshold = 0.9
Mapping crossover	Equal-sized blocks, threshold = 0.1
Population size	25 genotypes, 25 mappings (50 for traditional)
Fitness Cases	Heart: 14 attributes, the last specifies class (0 or 1) Breast: 10 attributes, the last specifies class (0 or 1)
Fitness	Raw correct classifications
Objective	Highest classification accuracy possible on test set.
Learning rate	0.1
Noise threshold	0.95

7.2 Interpretation of Instructions

Each genotype individual for the Medical problems consists of up to 320 bits and 4 registers to store sub results. The first segment of an instruction in the Standard Adaptive Mapping GP or Huffman mapping PAM DGP identifies one of the ten possible operators, where the length of this function identifier is dictated by Huffman's algorithm. More emphasis is generally placed on an operator by having a shorter identifier. In the Traditional (linear) GP algorithm and the Redundant mapping PAM DGP, the function identification segment of the instruction is of a fixed length (4 bits) where the integer representation of the bits in this segment is simply used to specify one of the ten operators. The binary representations of the integers wrap back to the first operator after the final operator for the default encoding of the Traditional GP.

A single 'flag' bit following the function identifier determines where the operand to the right of the operator is loaded from: either the operand is loaded from the registers, or the operand is loaded from one of the variables from the current fitness case. In the case where the operand is loaded from the registers, the integer representation of the two bits following the flag bit specify one of the 4 registers as the destination register, and the last two bits specify one of 4 registers as the source register. In the case where the operand is loaded from the current fitness case, the two bits following the flag bit specify one of the 4 registers as the destination register. The integer representation of the last following bits specify the one of the variables from the current fitness case to be loaded into the destination register, where the binary representations of the integers wrap back to the first attribute after the last attribute. The interpretation of an instruction is shown

below in Figure 7.1 for each of the four GP algorithms (Traditional GP, Standard Adaptive Mapping GP, Huffman PAM DGP, and Redundant PAM DGP).

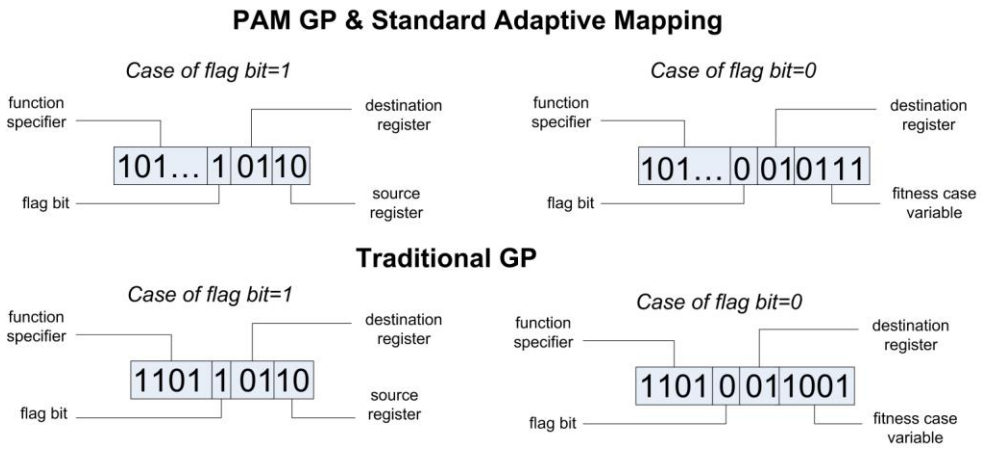


Figure 7.1. Parsing of instructions for the Medical classification problems.

7.3 Medical Classification Performance Results

The classification accuracy of the Heart and Breast classifiers after 30 000 and 50 000 rounds of training, respectively, over 50 independent trials are shown in Figures 7.2 and 7.3, respectively. Results describe performance for Traditional (linear) GP (Traditional), the Adaptive Mapping algorithm (Standard), PAM DGP using Huffman encoding (Huffman), and PAM DGP using adaptive redundant mappings (Redundant). The results are based on four-fold cross-validation, so the median and spread shown in the boxplot correspond to the mean accuracy across all four unique test sets over the 50 trials. Each box indicates the lower quartile, median, and upper quartile values. If the *central* notches of two boxes do not overlap, the medians of the two groups differ at the

0.95 confidence interval. Points represent outliers to whiskers of 1.5 times the interquartile range.

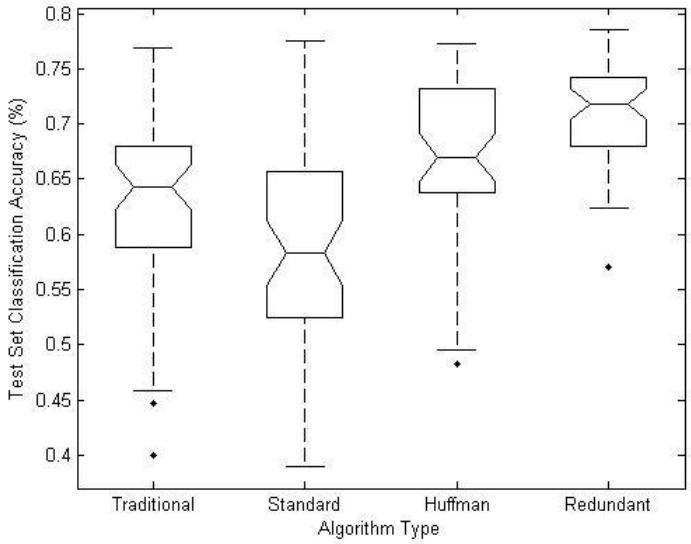


Figure 7.2. Boxplot of mean classification accuracy for the Cleveland Heart data set over 50 trials using four-fold cross-validation. Each partition was 75% training, 25% test. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times interquartile range.

It is evident from Figure 7.2 that the adaptive redundant mapping outperforms the Huffman mapping in PAM DGP at the 0.95 confidence interval, as well as outperforming all other algorithms examined at the 0.95 confidence interval (there is no overlap between Redundant's notch and the other algorithms). In fact, the redundant mapping also boasts the best median, general spread of data, and best accuracy achieved (note top of upper whisker). The Standard (Original) Adaptive Mapping algorithm of Margetts and Jones is the worst performer of all the algorithms at the 0.95 confidence interval, as well as when considering both median and general spread of the data.

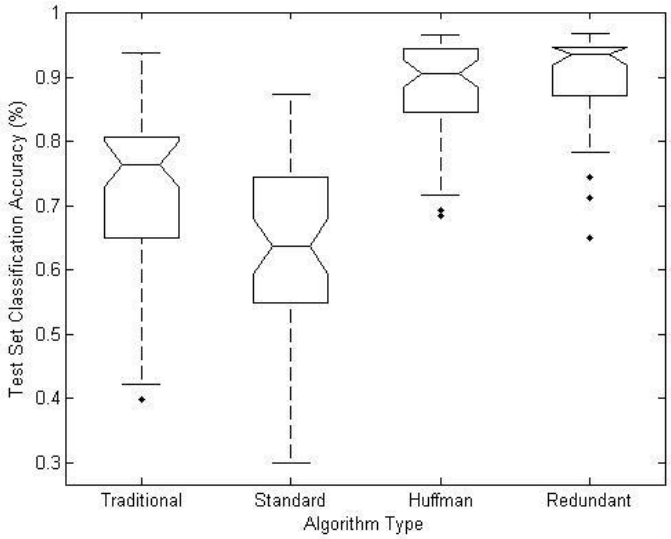


Figure 7.3. Boxplot of mean classification accuracy for the Wisconsin Breast data set over 50 trials using four-fold cross-validation. Each partition was 75% training, 25% test. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times interquartile range.

In the case of the easier Breast data set, Figure 10, while redundant mappings do not actually outperform Huffman mappings at the 0.95 confidence interval, the redundant mapping does outperform the Huffman mapping when considering median and general spread of the data (two rightmost box plots). It is certainly the case that the redundant mapping is not outperformed by the Huffman alternative in any respect. The redundant mapping, however, outperforms the two other algorithms at the 0.95 confidence interval. As was the case for the Heart Disease data, the Standard Adaptive Mapping, followed by Traditional GP, had the worst performance out of all four algorithms given 0.95 confidence interval, median, and general spread of the data. The PAM DGP implementations thus clearly outperform the other GP alternatives in both the Breast and Heart medical classification domains, with the redundant mapping being preferable.

7.4 Function Set Analysis

While redundant mappings are shown to outperform other algorithms, it remains to be shown that the redundant encoding is trimming the members of the function set to yield this improved performance (recall that Huffman encoding cannot trim the function set). It is also of interest to see whether or not there is any significance to the particular functions chosen by the redundant mappings to remain as useful members of the function set. The average number of unique function set members (unique operators) used in the classification solutions to the experiments described above is given below in Figures 7.4 and 7.5.

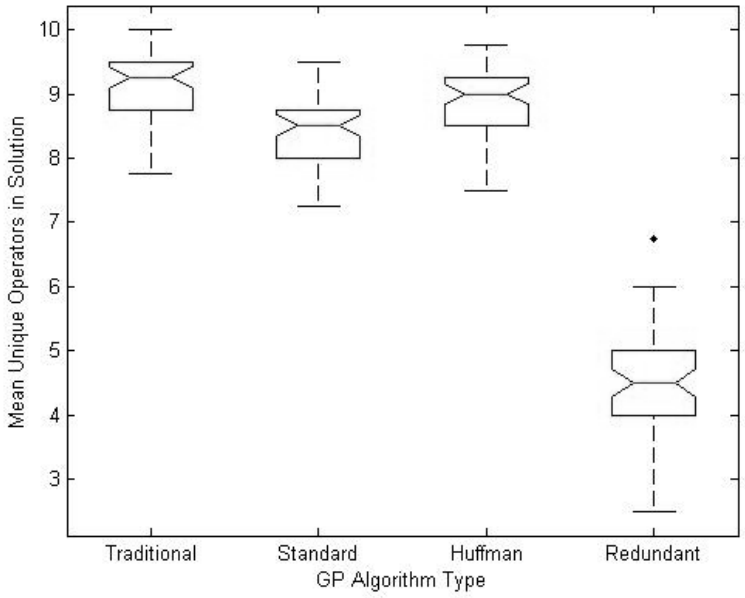


Figure 7.4. Boxplot of mean number of unique operators used in the classifier for the Cleveland Heart data set over 50 trials using four-fold cross-validation. Each partition was 75% training, 25% test. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times interquartile range.

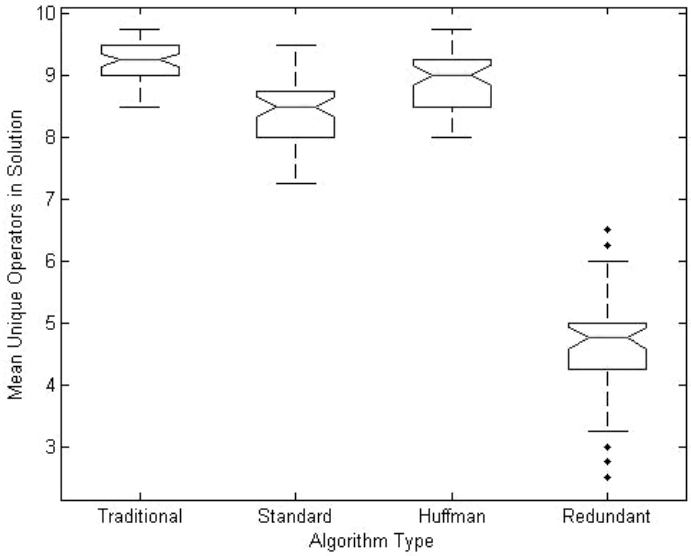


Figure 7.5. Boxplot of mean number of unique operators used in the classifier for the Wisconsin Breast data set over 50 trials using four-fold cross-validation. Each partition was 75% training, 25% test. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times interquartile range.

It is evident in both Figures 7.4 and 7.5 that the redundant mapping provided a considerable reduction to the size of the function set used in the evolved solutions. In both datasets, the highest outlier (most symbols of the function set) is still below the lowest number of function set symbols used in all other algorithms. In some cases, in both classification problems the average over the 4 partitions was fewer than 3 operators used in a solution. The redundant mapping is definitely improving classification performance (beyond any algorithm using Huffman encoding or Traditional GP) by trimming the function set. Part of the contribution to the performance may also be rooted in appropriate function emphasis, to which we now turn.

In terms of the functions chosen to remain in the function set, the results are shown in Figures 7.6 and 7.7 below for Heart and Breast data, respectively. The

Standard Adpative Mapping DGP has been dropped from Figures 7.6 and 7.7 for clarity, since it also uses Huffman encoding just like one of the implementations of PAM DGP and was shown in Section 7.3 to produce poor classifiers compared to it competitors.) We can see in Figure 7.6 that for the Heart classification problem that the Redundant mappings produce a statistically significant difference in operators for solution composition at the 99% confidence interval for every operator when compared to Huffman. Thus, the redundant mappings for the Heart problem not only tend to use less operators in a solution on average (Figure 7.4), but the operators it chooses for a solution are radically different than the Huffman encoding. We can also see that the Redundant mapping makes operator set choices, to its advantage, that are significantly different than Traditional GP: Figure 7.6 indicates that the Huffman mapping simply produces a very even distribution of the available operators across all solutions, which is not, as we have seen in previous fitness analyses, particularly advantageous. The Huffman mapping has an even more uniform distribution of function set members than the Traditional GP! In contrast, the Redundant mapping favors a distinctive combination of operators, using MULTIPLY, BASE 10 LOG, and NATURAL LOG to create its superior classifier (all significant at the 0.99 confidence interval).

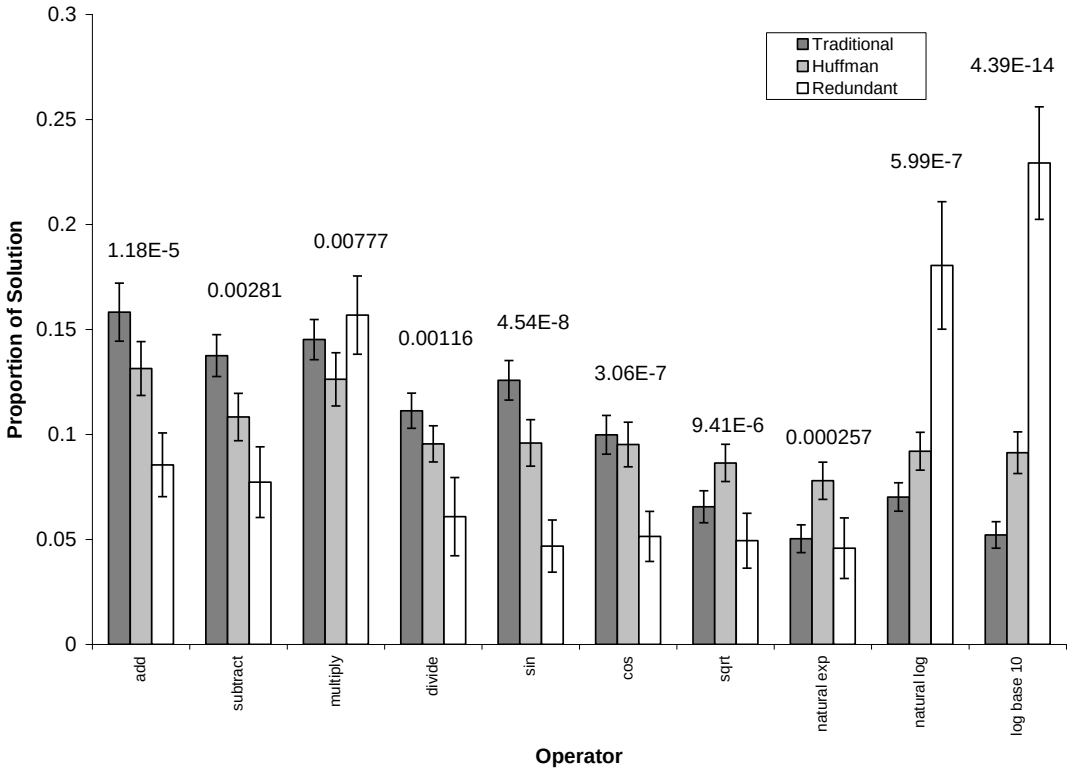


Figure 7.6. Mean operators as a proportion of total solutions over 50 trials for the Cleveland Heart Problem, population 50. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. P-values corresponding to Huffman and Redundant mappings are displayed above each set of data points.

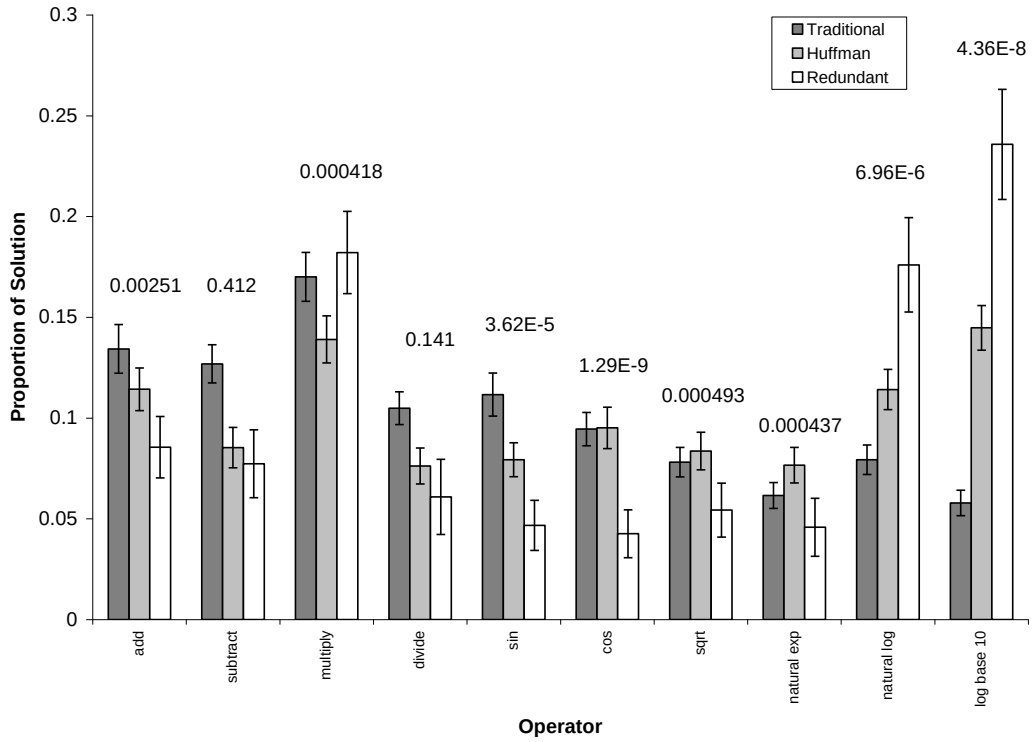


Figure 7.7. Mean operators as a percentage of total solutions over 50 trials for the Wisconsin Breast Problem, population 50. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. P-values corresponding to Huffman and Redundant mappings are displayed above each set of data points.

Figure 7.7 for the Breast classification data again shows a comparatively uniform utility of function set symbols when using a Huffman mapping, compared to Traditional GP or the Redundant mappings. The classifier produced using the redundant mappings has again favored the function set members multiplication, base 10 log, and natural log (again, all at the 0.95 confidence interval). We can see that for both classification problems, the adaptive redundant mapping has led to reduced function set size and greater emphasis of particular function set symbols to produce higher performance classifiers. It is also interesting that across both of these classification problems that the

redundant mapping implementations have chosen to heavily emphasize the same subset of three particular symbols to generate higher fitness classifiers.

7.5 Classification Problems Summary

This chapter describes classification problem domains where the benefits of using the Adaptive redundant mapping encoding scheme in PAM DGP are quite definitive. On the Heart benchmark medical classification problem, the redundant mapping was shown to produce definitively better classifiers than all other GP-based algorithms for the metrics of median, general spread of data, best result, and it was superior at the 0.95 confidence interval. On the Breast medical classification problem, the redundant mapping has the best median and general spread of data compared to all other algorithms. The Standard Adaptive Mapping, followed by Traditional GP, is the worst performer for both problems. Thus, PAM DGP in general is found to be a valuable framework for improving on the performance of existing GP algorithms on Medical classification problems featuring comprehensive function sets. Furthermore, given the use of PAM DGP, redundant mappings provide better classifiers using all (Heart) or a number (Breast) of metrics and in no case are outperformed by Huffman encoding. Thus, Redundant PAM DGP is the obvious choice for the Medical classification domains presented here.

This chapter also demonstrates that the redundant mapping is providing higher accuracy solutions by trimming the function set. An examination of the particular operators chosen by the end solution of each algorithm also provided the interesting result that the redundant mapping chooses the same three functions to construct the

classifiers in both the Heart and Breast problem instances. The next chapter will further explore the benefits of the Adaptive redundant mapping in PAM DGP over existing algorithms through its application to difficult sequence learning problems involving recursive solutions.

Chapter 8. Using PAM DGP to Evolve Recursive Functions using Machine Language Instructions

8.1 Problem Definitions and Parameterization

The results in this chapter examine the abilities of different adaptive mappings and algorithms to effectively choose the appropriate members of a function set consisting of *general* (not implicitly recursive) instructions to efficiently discover solutions to recursive problems, i.e., instructions explicitly implementing loop or subroutine calls are not provided. Huelsbergen has published several papers on evolving machine language solutions, using generic instructions so as to automatically produce iteration for applications including recursion, sorters, the parity problem, and most recently custom compilers [37-41]. The machine language-based function set for his work on recursion [40], which we adopt, has the property that it does not require domain-specific recursive operators to be introduced in order to create programs that generate a general recursive solution.

No approaches known to Huelsbergen (or this author) prior to the publication of [40] in 1997 (as stated in said work) were able to generate recursive solutions without introduction of operators to the function set in order to explicitly enable recursion. Works mentioned by Huelsbergen include early works of Koza [49, 51] and Handley [33], where both authors introduce specialized operators into their function sets and thus avoid the challenge of automatically synthesizing recursion. Koza has recently continued to implement more specialized functions (automatically defined functions, or ADFs) to perform recursion in [50]. Other authors such as Brave [17] and Yu [100, 102] have

opted in the past to evolve recursive programs by including the name of the function in the function set. Similarly, Wong and colleagues [98, 99] have implemented GP systems using logic grammars capable of recursion. Whigham [92] has used directed mutation operators to evolve a recursive function, but operators are both problem specific and incorporate knowledge of the solution. Yu has also presented an interesting technique that uses implicit recursion via higher order functions to achieve recursion without explicit recursive calls [101, 104]. In her solution, the code content of a recursive program is passed as an argument to the higher-order function that iteratively applies the code. While avoiding explicit recursion calls, the recursive mechanism is built into the higher order function and is thus not automatically generated. (The use of higher order functions does have the nice benefit that it implicitly provides a termination mechanism for recursion.) It seems that, as of this writing, Huelsbergen has been the only researcher to attempt automatic generation of recursion using only a 'low level' machine language-based function set. In contrast, the focus of other researchers has been the issue of measuring good "semantics" in recursive solution program structures and handling non-terminating recursive cases [101, 104]. His concern (and that of this chapter) is to actually discover recursive solutions using a function set that does not imply recursion in any way. This chapter does address the issue of semantics through a simple metric (correct sequence output length prior to program termination) to indicate semantic goodness of solutions. The termination issue of recursive solutions is handled in the usual way—by reaching a maximum number of program steps executed (this method is used in most of the literature on recursion, with the notable exception of [101, 104]).

Huelsbergen's function set is generic such that it consists only of instructions for primitive register manipulation, conditional and unconditional branching, arithmetic operators, and generation of an output stream. Each individual consists of a program with a number of external registers, and internal state trackers including a program counter (*PC*) and a flag (*Flag*). *Flag* corresponds to the last execution of a special comparison instruction (*Cmp(*R_{source}*,* R_{dest}*))* that returns one of the values {*greater, less, equal*}; it serves as a basis on which to perform conditional branching. The program counter is an integer that points to the instruction to be currently executed; branching (*jump*) instructions cause the PC to point to the target of the branch, while remaining instructions cause PC to point to the following instruction.

The function set is designed to correspond to a virtual register machine (VRM). The *Output* function places an integer from a register on the output stream *Stdout*; if no output is generated by an individual the *Stdout* stream contains no values (no output is produced by the program). The function set uses the CMP function to compare registers to generate a value for the flag state, and MOV copies a value from a source register to a specified destination register. J is an unconditional branch instruction, whereas JL, JG, and JE branch conditional on the value of *Flag*. The offset of the jump operation is relative to the current location of the program counter, with a negative offset being a backward branch and a positive offset a forward branch. The definition of the branching functions corrects a branch to a location PC < 0 to PC = 0, and a branch past the end of the program (PC > *n*-1 where *n* is the number of instructions in the program) to the index of the final instruction (PC = *n*-1, or termination). SET and CLEAR instructions reset register values to 1 or 0, respectively, while INC and DEC increment and decrement,

respectively, the values in their register arguments. The arithmetic operators ADD, SUB, MUL, and DIV perform their operation between a source and destination register with the result left in the destination register. The arithmetic operators DIV and SUB are protected against divide-by-zero and underflow exceptions, respectively. NEG negates the value in its specified register. Huelsbergen adds a NOP function which performs no operation, which we opt to omit because it is not a useful function for a final solution and its omission provides an appropriate number of function set members (16) for the default encoding of the Traditional GP implementation. The pseudocode for the function set is given in Figure 8.1 below.

```
Out(Rsource) {
      PC = PC + 1;
      Write(Stdout, Rsource);
}

Neg(Rsource) {
      PC = PC + 1;
      Rsource = 0 - Rsource;
}

Mov(Rdestination, Rsource) {
      PC = PC + 1;
      Rdestination = Rsource;
}

Set(Rsource) {
      PC = PC + 1;
      Rsource = 1;
}

Clear(Rsource) {
      PC = PC + 1;
      Rsource = 0;
}

Inc(Rsource) {
      PC = PC + 1;
      Rsource = Rsource + 1;
}
```

```
Dec(Rsource) {
      PC = PC + 1;
      Rsource = Rsource - 1;
}

Add(Rdestination, Rsource) {
      PC = PC + 1;
      Rdestination = Rdestination + Rsource;
}

Sub(Rdestination, Rsource) {
      PC = PC + 1;
      Rdestination = Rdestination - Rsource;
}

Mul(Rdestination, Rsource) {
      PC = PC + 1;
      Rdestination = Rdestination * Rsource;
}

Div(Rdestination, Rsource) {
      PC = PC + 1;
      Rdestination = Rdestination / Rsource;
}
```

```
Cmp(Rdestination, Rsource) {
      PC = PC + 1;
      If (Rdestination < Rsource) Flag = less;
      Else If (Rdestination > Rsource) Flag = greater;
      Else Flag = equal;
}

J(offset) {
      PC = min(max(0, PC + offset), NoOfInstructions);
}

Jl(offset) {
      If (Flag == less)
       PC = min(max(0, PC + offset), NoOfInstructions);
      Else PC = PC + 1;
}

Jg(offset) {
      If (Flag == greater)
       PC = min(max(0, PC + offset), NoOfInstructions);
      Else PC = PC + 1;
}

Je(offset) {
      If (Flag == equal)
       PC = min(max(0, PC + offset), NoOfInstructions);
      Else PC = PC + 1;
}
```

Figure 8.1. Function set of 16 instructions for creation of generalized recursive solutions. Sub and Div are protected against underflow and divide-by-zero exceptions.

In [40], Huelsbergen investigates four integer sequence problems: a sequence of squared numbers, cubed numbers, and the factorial and Fibonacci sequences. We focus on the more difficult and naturally recursive Factorial and Fibonacci sequences in this work, which are defined with the same initial values as Huelsbergen. We then increase the difficulty of determining a program for the sequence by increasing the depth of the recursion to third order. The functions that are thus chosen to produce sequences for this investigation are Factorial (*fact*), Fibonacci (*fib*), third order Fibonacci (*fib3*). The function definitions, including base cases, are

$$fact(x) \equiv \begin{cases} 1 & \text{if x = 0} \\ x \cdot fact(x) & \text{otherwise} \end{cases} \tag{8.1}$$

$$fib(x) \equiv \begin{cases} 1 & \text{if x = 0 or x = 1} \\ fib(x-2) + fib(x-1) & \text{otherwise} \end{cases} \tag{8.2}$$

$$fib3(x) \equiv \begin{cases} 1 & \text{if x = 0, x = 1, or x = 2} \\ fib3(x-3) + fib3(x-2) + fib3(x-1) & \text{otherwise} \end{cases} \tag{8.3}$$

The fitness evaluation scheme is reproduced from [40] as described by Huelsbergen. The functions are used to generate the first ten values of each sequence (a ten value function prefix), which serves as the test case. The *Stdout* stream generated by the `OUT` instruction is matched against the ten values of the test case using the following fitness function:

$$fitness(p) \equiv \sum_{i=0}^{l-1} \left| s_i - f(i) \right| \cdot scale(i) \tag{8.4}$$

where p is the program in the form a binary string, l is the maximum length of the recursive sequence (10 in these experiments), $f(i)$ is the value of the recursive function for integer i, and $scale(i)$ is defined as

$$scale(i) \equiv \begin{cases} S_{\max} & \text{if } f(i) = 0 \\ S_{\max} / f(i) & \text{otherwise} \end{cases} \tag{8.5}$$

where $S_{max} = \max\{f(0), \ldots, f(l-1)\}$ for the recursive sequence defined by f. The sequence $\{s_0, \ldots, s_{l-1}\}$ is the first l values of *Stdout*, if the output contains at least l values. If it does not, the $j < l$ values *Stdout* contains (that is, $\{s_j, \ldots, s_{l-1}\}$) are given the value S_{max}. The fitness function effectively measures amplified raw error, so lower fitness is better.

The full summary of the GP parameters used in these experiments is given below in Table 8.1. As usual, a population size of 50 (25 genotypes and 25 mappings) is used, with equal crossover and mutation rates (0.9 and 0.5, respectively) to allow high levels of exploration and a moderate introduction of new material to explore in both populations. Learning rate was kept the same as other experiments (0.1), but the noise threshold was lowered to 0.8 from previous experiments to prevent trapping of algorithms in local optima. Since Huelsbergen's results indicated that a larger number of tournament rounds would likely be necessary to generate recursive solutions compared to previous experiments, each trial was allowed to run for 500 000 rounds.

Table 8.1. Recursive Problems parameterization.

Tournament Style	Steady State, 4 individuals for each round
Maximum Rounds	500 000
Function Set	OUT, SET, CLEAR, INC, DEC, NEG, MOV, ADD, SUB, MUL, DIV, CMP, J, JL, JG, JE
Genotype structure	Instruction sequence with 4 registers; 320 bits; PC and Flag for internal program state
Mapping structure	160 bits (10 bits per function set member)
Genotype mutation	XOR mutation, threshold = 0.5
Mapping mutation	Point mutation, threshold = 0.5
Genotype crossover	Equal-sized blocks, threshold = 0.9
Mapping crossover	Equal-sized blocks, threshold = 0.9
Population size	25 genotypes, 25 mappings (50 for traditional)
Test Case	First 10 values of function sequence
Fitness	Scaled raw error between *Stdout* and test case
Objective	*Stdout* produces the first ten natural numbers specifed in the test case (fitness/error of 0).
Learning rate	0.1
Noise threshold	0.8

8.2 Interpretation of Instructions

Genotype individuals take the form of virtual register machines (VRMs) including a set of instructions (program), four external registers, the program counter (PC), and flag variable to track internal state as outlined above. Each genotype individual consisted of 320 bits for the genotypes. Since there were 16 members of the function set, the mapping individuals consisted of 160 bits to provide ten bits per function. As was the case in Huelsbergen's experiments in [40], the program in each genotype individual terminates after running all instructions (PC = n-1 for n instructions with indices 0 to n-1) or after the execution of 100 steps.

The interpretation of each instruction begins with parsing the starting bits to identify a member of the function set. As usual, the number of bits required for this step is variable for all implementations using the Huffman encoding (Standard Adaptive Mapping GP and Huffman mapping PAM DGP) and is fixed for those implementations not using Huffman (Traditional GP and adaptive redundant mapping PAM DGP). There are three types of functions in the set with respect to number and type of arguments, each requiring a separate interpretation scheme following the initial identification of the function. The first type of function takes two arguments, specifying a destination and source register. The functions included in this group are {MOV, ADD, SUB, MUL, DIV, and CMP}. In this case, the two bits following the function bits are the binary representation of the integer specifying one of four registers as the source, with the following two bits specifying the destination register. The second type of function takes a single argument determining value of the register to which the action specified by the

function is applied. Functions using this format include {OUT, SET, CLEAR, INC, DEC, and NEG}. The last type of function accepts a single argument as well, but it is an offset indicating the location to which the program counter (PC) should currently point. Only the branching instructions {J, JL, JG, and JE} follow this format. In this case, the five bits following the function specifier are interpreted as an integer, and that is taken as the offset. The first of those five bits is a sign indicator: If it is set to 0, the offset is negative; if it is set to 1, the offset is positive. The remaining four bits constitute the absolute binary representation of the offset, so the offset can take one of sixteen integer values in the interval [0, 15]. The interpretation of the instructions is depicted below in Figure 8.2.

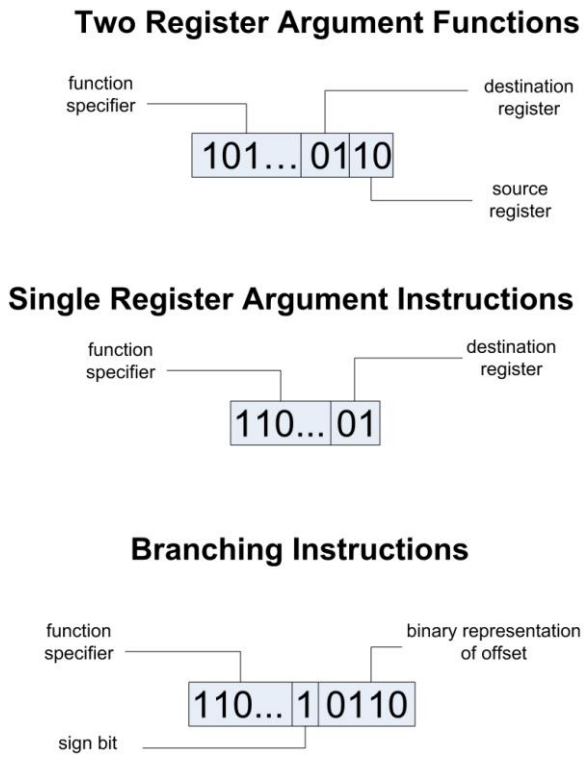


Figure 8.2. Parsing of instructions for the recursive problems.

8.3 Recursion Performance Results

8.3.1 Terminology and Classification of Solutions

In [40], Huelsbergen compares the abilities of random search (Random), traditional genetic programming using solely the crossover operator (XO), exhaustive iterative hill climbing (EIHC), and a hybrid system of his own design that uses both techniques (XO-EIHC). He found that the simple genetic search (XO) performed the best out of all algorithms for the factorial function, but the more sophisticated EIHC and XO-EIHC algorithms outperformed the other algorithms definitively for the Fibonacci series: 10 solutions were found given the first of either 10 solutions or 5×10^7 evaluations for each of EIHC and XO-EIHC, and no solutions were found for Random and XO. Sample solutions from the XO-EIHC algorithm were then shown to produce general solutions to the recursive problems through use of an infinite loop constructed from the branching functions. Our analysis of the recursive functions will examine the number of solutions generated by each algorithm, as well as an in depth analysis (and associated definitions) of solution generality and quality.

In this section we compare the efficiency, solution content, and solution quality of Traditional GP (*Traditional*), the Standard Adaptive Mapping DGP of Margetts and Jones (*Standard*), PAM DGP with Huffman encodings (*Huffman*), and PAM DGP with Adaptive Redundant encodings (*Redundant*). Some discussion of the recursive solutions produced by the algorithms covered in this work is in order before proceeding with the analysis of the results.

Following Huelsbergen [40], a *solution* is said to have been located when the output stream of an individual's program produces the first ten digits of the required

sequence (which is the test case).[6] Given this definition of *solution*, if the program produces incorrect digits or no digits after getting the initial ten digits correct, it is still technically a solution. In this work, a program is considered a *general solution* if and only if *both* all the members of the sequence of length $l \geq 10$ generated by the output are correct *and* if the program were permitted to run beyond the maximum number of steps (100 in [40] and these experiments), then the program would continue to correctly generate the correct members of the sequence. All solutions (programs that generated the first ten members of the recursive sequence correctly) in these experiments were inspected by hand for generality. In practice, given the function set for these problems, a solution could only be general if it included an appropriate instruction sequence using a reverse branch (jump instruction with negative offset) at the end of the sequence. Furthermore, the repeated sequence would have to include appropriate manipulation of register contents and an output to the *Stdout* stream in its body such that the correct outputs were produced. The results focus on the ability of the algorithms to produce not just solutions, but *general* solutions.

8.3.2 The Factorial Function

The first recursive function we examined was the factorial sequence (Equation 8.1), which is a first order recursive function. That is, each iteration of the recursive function only references the value produced by the previous recursive step. In this

[6] To be precise, a small number of the subset of solutions that produced ten correct members of the sequence actually actively generated only nine correct sequence members. This was accomplished by taking advantage of a loophole in Huelsbergen's scaling aspect of the fitness function (Eq. 8.5). If the output does not contain at least $l=10$ values, the $j < l$ values *Stdout* contains (that is, $\{s_j, \ldots, s_{l-1}\}$) are given the value S_{max}. S_{max} for the functions investigated is the largest—and last—number in each sequence, automatically setting *Stdout*'s last member to the correct value.

respect, the Factorial problem is the simplest of the recursive functions considered. As mentioned earlier, Huelsbergen found that it was most efficiently solved by simple genetic search using only two-point crossover rather than his more sophisticated search techniques [40]. We similarly found that the less complex algorithms generated more solutions: given 50 independent trials, all trials for Traditional, Standard, and Huffman PAM DGP solve the factorial problem, as does 33 trials of Redundant PAM DGP. The number of solutions found by each algorithm are shown in Figure 8.3 immediately below. In the case of the factorial problem, every solution for all algorithms was general.

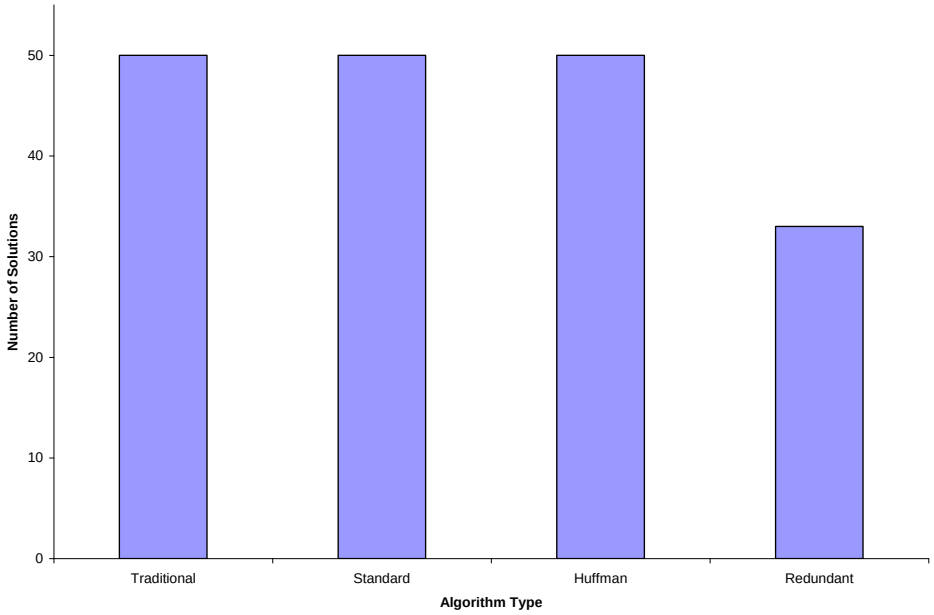


Figure 8.3. Number of solutions produced by each algorithm over 50 independent trials for the factorial function.

The tournament round when the solution was located for each solution in 50 independent trials is given in Figure 8.4 for the factorial problem. Each box indicates the lower quartile, median, and upper quartile values. Central notches indicate the 0.95 confidence interval, with points representing outliers to whiskers of 1.5 times the

interquartile range. Given the overlap of the notches for the boxplots, there is actually no statistical difference at the 0.95 confidence interval in the median round at which a solution is found for any of the four algorithms. Huelsbergen's hybrid algorithm had a mean of 5.55 x 10^6 evaluations required per solution (over 9 solutions) for the factorial function, while Redundant PAM DGP had a mean of only 6.81 x 10^4 evaluations (4 evalutions per round) required per solution (over 33 solutions).

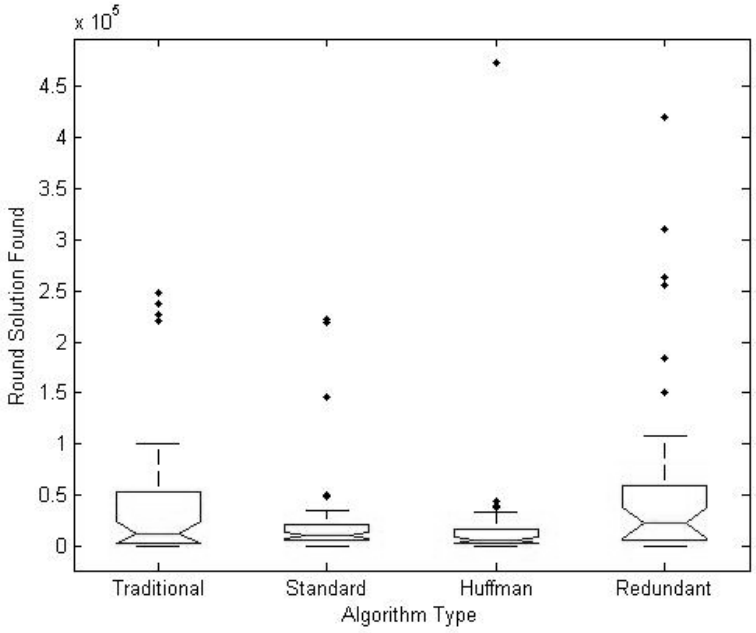


Figure 8.4. Tournament round at which a solution to the factorial problem was located for all solutions found over 50 independent trials for all algorithms.

While Redundant PAM DGP did not produce as many solutions as the other GP algorithms for this simple recursive function, it outperforms the other algorithms on solution quality. The programs that are of interest are those that have truly discovered recursive solutions, and are thus *general*. One way to measure the quality of these general solutions is to examine how many members of the function's sequence the solution can produce before it reaches the program step limit. That is, efficiency of the

program at generating the sequence is measured. The efficiency of sequence generation is an important measure: If the body of the loop(s) that produce the sequence contain junk code (introns), program steps will be (at best) wasted if the junk code is innocuous in so far as it does not disrupt the production of the sequence. A loop with innocuous junk code will produce a less lengthy sequence. In fact, introns must be innocuous in general solutions or the solutions would not be able to generate the repeated sequence indefinitely. Efficiency also reflects that the algorithm may be generating multiple outputs per iteration to avoid wasting steps on the jump instructions. Thus, the higher the value of the correct number of sequence members generated, the lower the content of junk code within the program loop(s) and/or the more efficient the loop(s) contents. The number of sequence members produced is thus a simple and informative measure of the quality of general recursive solutions. The number of sequence members produced by the *general* solutions to the factorial problem of each algorithm is shown in Figure 8.5.

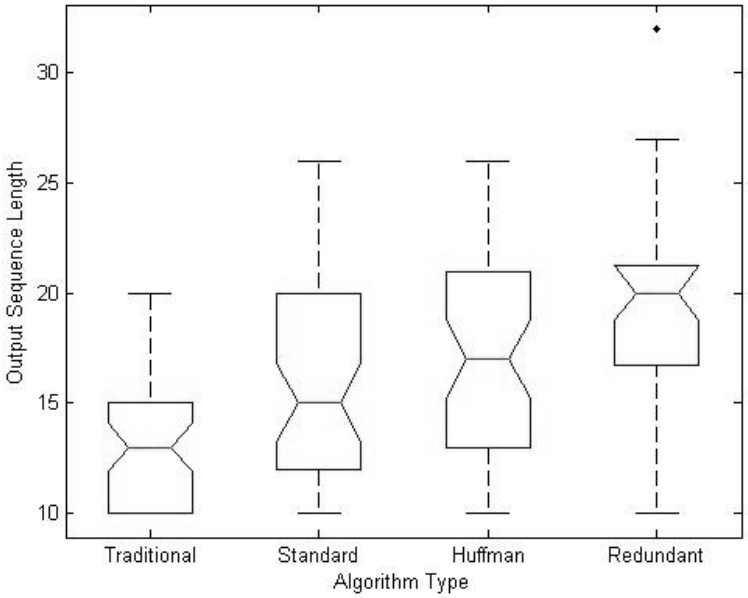


Figure 8.5. Number of sequence members output by the general solutions to the factorial problem produced over 50 independent trials by all algorithms.

It is evident that the Redundant PAM DGP algorithm produces the longest sequences among its general solutions at the 0.95 confidence interval compared to all other algorithms. Given the aim of discovering a program to produce quality general recursive solutions, rather than sheer quantity of solutions regardless of quality or even generality, Redundant PAM DGP clearly provides the best results on the factorial problem. The best general solution produced the first 32 members of the factorial sequence, and can be seen as the upper outlier for Redundant PAM DGP in Figure 8.5. The program code for the individual is given in Figure 8.6. This solution contained no introns. The loop responsible for the indefinite repeated production of the series is italicized. Any instructions that are not reached by the program counter (instructions that are never read by the hypothetical interpreter) are not displayed. Instruction addresses are enumerated on the left of each instruction to help the reader better interpret branching commands.

```
 0 INC Reg 2
 1 OUT Reg 0
 2 OUT Reg 1
 3 OUT Reg 2
 4 INC Reg 1
 5 INC Reg 2
 6 MUL Reg 1 * Reg 2
 7 OUT Reg 1
 8 INC Reg 2
 9 MUL Reg 1 * Reg 2
10 OUT Reg 1
11 INC Reg 2
12 MUL Reg 1 * Reg 2
13 OUT Reg 1
14 INC Reg 2
15 MUL Reg 1 * Reg 2
16 OUT Reg 1
17 J(to 8) using offset -9
```

Figure 8.6. Program code for the individual that produced the longest factorial sequence. Instructions that constitute the loop are italicized.

In the solution above, instructions 1 to 7 generate, via sequential non-looping instructions, the first four values of the factorial series (1, 1, 2, 6) and thus set up the base case (first value) prior to entering the loop. Instruction 8 begins the loop body that contains three consecutive INC, MUL, OUT sequences that maintain the function's *x* and *x-1* values in registers 2 and 1, respectively. The loop efficiently uses all instructions in its body to output three members of the factorial solution with each iteration. This solution demonstrates the nature of the efficiency and generality of the solutions produced by Redundant PAM DGP, as quantified in Figure 8.5.

8.3.3 The Fibonacci Series

We now move to measuring the capability of the algorithms on a more challenging recursive problem: the Fibonacci series. The Fibonacci series uses, by definition, second order recursion. In other words, the current value of the function (with

the exception of the base cases, of course) depends on the values of the two previous recursive steps. Huelsbergen found that only his more sophisticated algorithms (EIHC and XO-EIHC) were able to produce solutions to the Fibonacci series; the other algorithms (XO and Random) produced no solutions given a limit of 5×10^7 evaluations. The number of solutions found by each algorithm we have been considering is provided below in Figure 8.7.

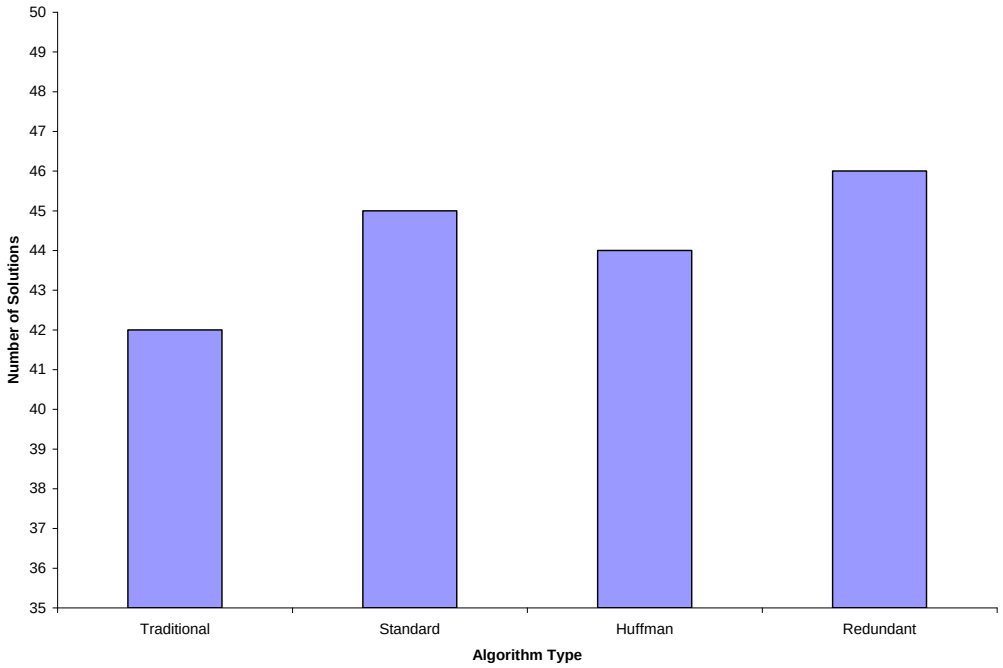


Figure 8.7. Number of solutions produced by each algorithm over 50 independent trials for the Fibonacci function.

Redundant PAM DGP produces the largest number of solutions (46), with Standard and Huffman PAM DGP producing comparable numbers of solutions (45 and 44, respectively). Traditional GP produced the least number of solutions (42). The boxplot for the tournament rounds at which a solution was located over 50 independent trials is below in Figure 8.8.

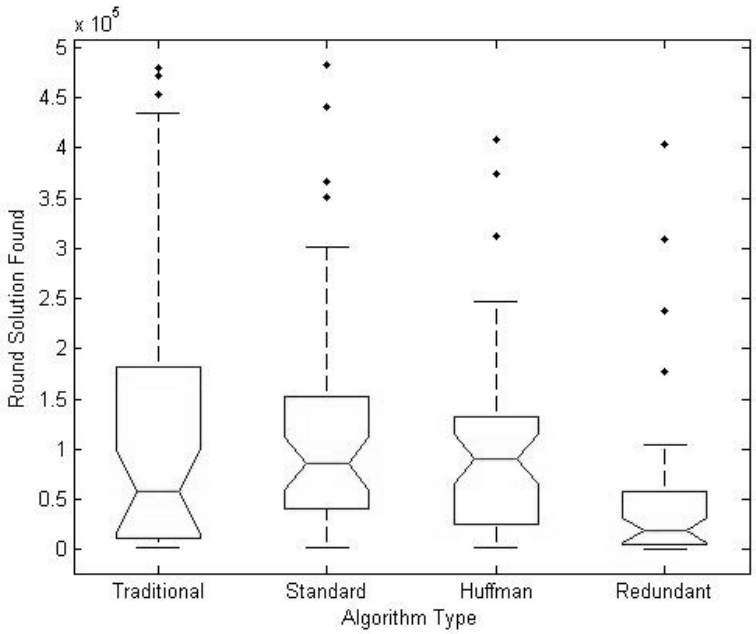


Figure 8.8. Tournament round at which a solution to the Fibonacci problem was located for all solutions found over 50 independent trials for all algorithms.

Redundant PAM DGP finds the Fibonacci series within fewer rounds than the Standard Adaptive Mapping and Huffman PAM DGP at the 0.95 confidence interval. Redundant PAM DGP also has a lower median than Traditional GP, but due to the large error level in the Traditional GP boxplot, the difference is not statistically significant. The spread of the Redundant PAM DGP boxplot also indicates that it solves the problem more consistently than any other algorithm. Huelsbergen's hybrid algorithm in [40] had a mean of 1.02×10^6 evaluations required per solution (over 10 solutions), while Redundant PAM DGP had a mean of only 2.12×10^5 evaluations required per solution (over 46 solutions) .

Considering the raw number of general solutions found, all algorithms actually generated comparable results. Redundant PAM DGP had 38 general solutions, Huffman PAM DGP had 42, Standard Adaptive Mapping found 43, and Traditional GP located 41. Despite having the lowest (but competitive) raw number of general solutions, Redundant

PAM DGP definitively generated the highest quality (most general) solutions. The sequence length of the solutions generated by each algorithm over 50 independent trials is shown in Figure 8.9.

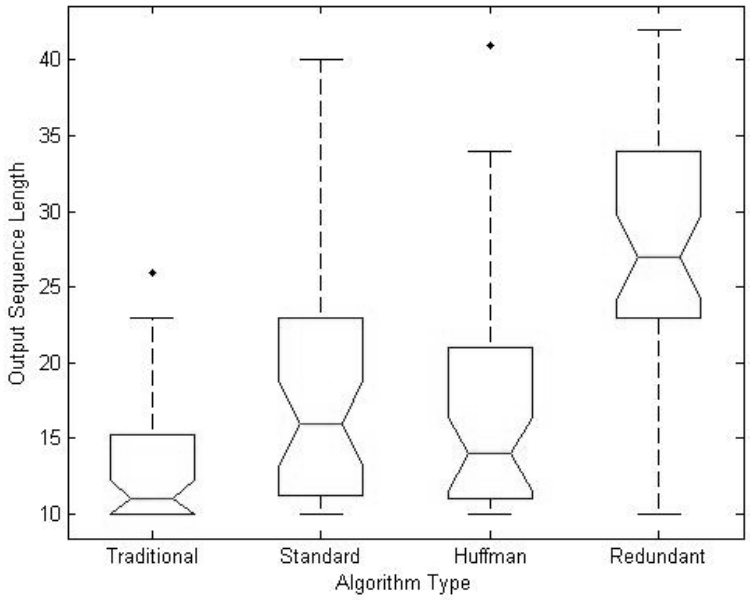


Figure 8.9. Number of sequence members output by the general solutions to the Fibonacci sequence over 50 independent trials by all algorithms.

Redundant PAM DGP, as was the case for the factorial problem, outperforms all other algorithms in terms of efficiency of solutions in generating the series. For the Fibonacci series, however, the degree to which Redundant PAM DGP outperforms the other algorithms is more considerable: The lower end of the interquartile range for Redundant PAM DGP's output length is above the top of the interquartile range for all other algorithms. It was noted that almost all of the solutions found by Traditional GP were general solutions (41 of 42 solutions); however, we can see in Figure 8.9 that Traditional GP achieved a median of only 11 sequence members. This means that Traditional GP was typically barely able to generate its minimum output length within its

solutions—its solutions are thus not efficient despite their generality. The median performance of the Standard Adaptive Mapping and Huffman PAM DGP were also significantly lower than Redundant PAM DGP, indicating that despite generating more general solutions, their solutions were also not as efficient. The longest solution Redundant PAM DGP generated was 42 members of the Fibonacci series, of which there were two distinct instances. The programs that produced these solutions are given in Figure 8.10. As before, any instructions of the individual's program that were never reached by the program counter (never interpreted or executed) are not displayed.

```
SOLUTION 1

 0 OUT Reg 3
 1 ADD Reg 0 + Reg 3
 2 OUT Reg 2
 3 ADD Reg 2 + Reg 0
 4 OUT Reg 0
 5 OUT Reg 2
 6 ADD Reg 0 + Reg 2
 7 J(to 3) using offset -4

SOLUTION 2

 0 ADD Reg 2 + Reg 1
 1 OUT Reg 3
 2 OUT Reg 0
 3 ADD Reg 1 + Reg 2
 4 OUT Reg 2
 5 OUT Reg 1
 6 ADD Reg 2 + Reg 1
 7 J(to 3) using offset -4
```

Figure 8.10. Program code for the individuals that produced the longest Fibonacci sequence. Instructions that constitute the loop are italicized.

In both of these solutions, there is a similar structure and neither solution includes any intron code in the body of the loop or otherwise. PAM DGP thus produces intron-free solutions to factorial and Fibonacci, whereas Huelsbergen's featured solutions for both functions in [40] contained introns. Both cases represent succinct, general recursive programs for generation of the Fibonacci series. Two of the first three instructions in each of the solutions establish the two required base case values, and the third performs a constructive addition instruction. Instructions 3 to 7 in both solutions comprise the loop that would indefinitely generate the Fibonacci series (in the absence of an upper limit of execution steps). Both loops generate two consecutive members of the series per iteration through a pair of addition and output instructions. Both of these solutions represent very efficient use of the available execution steps.

8.3.4 The 3rd Order Fibonacci Series

The final function we consider is the third order Fibonacci series as defined in Equation 8.3. The equation simply involves summing the results of the past three values in the series as opposed to the classic, second order, Fibonacci series where the previous two values in the series are summed to determine the current value. The number of solutions generated by each algorithm is shown below (Figure 8.11).

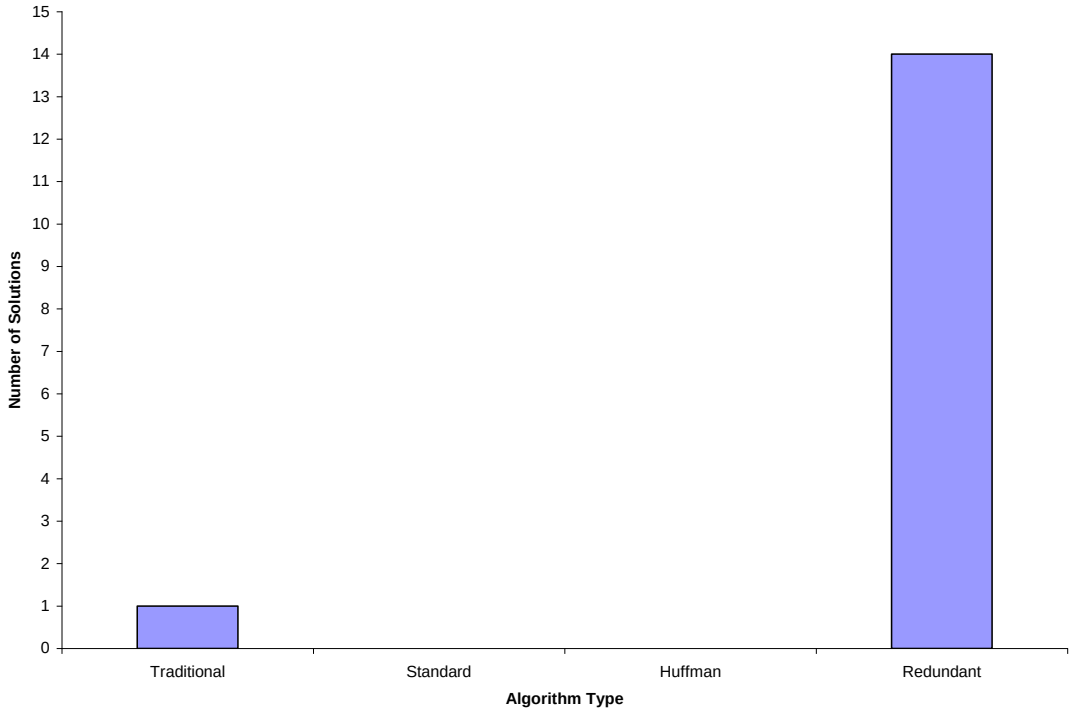


Figure 8.11. Number of solutions produced by each algorithm over 50 independent trials for the third order Fibonacci function.

Over 50 trials, neither Standard Adaptive Mapping nor Huffman-encoded PAM DGP produced any solutions. Traditional GP produced only one solution, and Redundant PAM DGP produced 14 solutions. Redundant PAM DGP is clearly better able to generate solutions to a recursive problem of this depth with a significantly higher degree of reliability than any other algorithm. The tournament round when the solutions were located is shown below in Figure 8.12; only algorithms that produced a solution are presented.

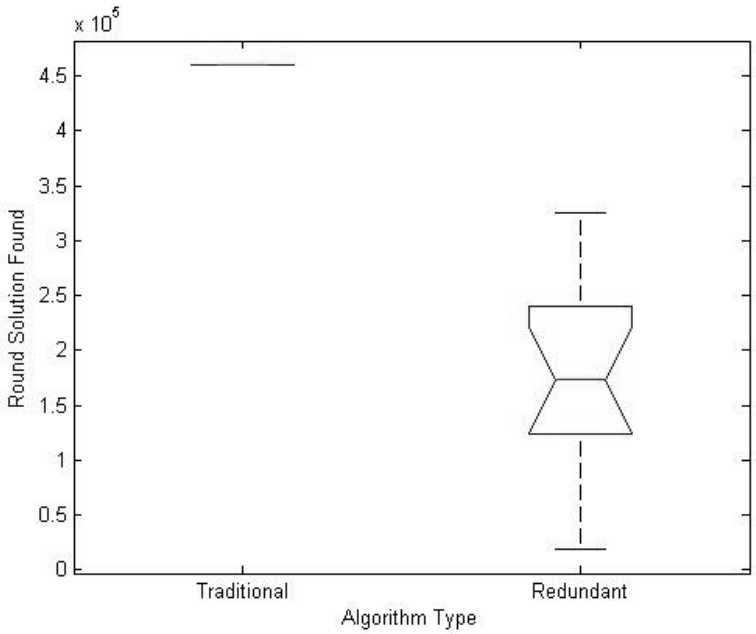


Figure 8.12. Tournament round at which a solution to the third order Fibonacci problem was located for all solutions found over 50 independent trials for Traditional GP and Redundant PAM DGP algorithms.

The single solution that was located by Traditional GP was found close to the maximum number of allowable tournament rounds. In comparison, Redundant PAM DGP located its solution much more efficiently across all of its 14 solutions. Only one *general* solution was found among all converging trials for both Traditional GP and Redundant PAM DGP, and it was one of the 14 solutions of Redundant PAM DGP and generated 25 members of the third order Fibonacci series. The program expressing that general solution is given below in Figure 8.13, with the associated values for the first iteration beside each instruction to aide the reader in comprehension of the solution. Only the instructions that were executed are displayed, and the solution contained no introns.

```
0 OUT Reg 0 value = 1.0
1 OUT Reg 2 value= 1.0
2 OUT Reg 0 value= 1.0
3 ADD Reg 3 + Reg 2 now= 2.0
4 ADD Reg 2 + Reg 0 now= 2.0
5 ADD Reg 0 + Reg 3 now= 3.0
6 ADD Reg 0 + Reg 3 now= 5.0
7 INC Reg 2 now= 3.0
8 OUT Reg 2 value= 3.0
9 ADD Reg 3 + Reg 2 now= 5.0
10 OUT Reg 3 value= 5.0
11 ADD Reg 3 + Reg 2 now= 8.0
12 J(to 4) using offset -8
```

Figure 8.13. Program code for the individual that produced the longest third order Fibonacci sequence in a general solution. Instructions that constitute the loop are italicized.

The methodology used by this solution is actually an interesting, less direct approach than simply adding the previous three values to generate the value for the current time step. The first four instructions generate the three required base cases by placing three 1.0s in the sequence and placing an initial value in Register 3. The loop actually causes repeated pairwise output of the values in Register 2 and 3 to produce all values following the base cases. Register 2, in addition to holding values to be output, helps Register 3 to generate its next sequence member two values in advance. That is, if Register 3 has output sequence member n_t (instruction 10), Register 2 adds the last member it output (n_{t-1}) to Register 3 in instruction 11, and then Register 2 adds the necessary difference to generate n_{t+2} (instruction 9) in the following iteration of the loop just prior to Register 3's output. Register 2 generates its next value following output in instruction 8 by having the correct difference to its next value added to it (instruction 4) from a subresult in Register 0 from a previous iteration of the loop (instructions 5 and 6),

along with an increment in the current iteration (instruction 7). Because Register 3 relies on Register 2, sequence members generated by Register 3 indirectly rely on all the instructions that contribute to Register 2. There is an indirect interwoven relationship among the instructions to create an innovative solution to the harder recursive problem.

8.4 Function Set Analysis

It has been demonstrated empirically in Section 8.3 that Redundant PAM DGP produces the most efficient general solutions over the factorial, Fibonacci, and third order Fibonacci recursive functions. This section investigates whether there was an underlying trimming of the function set to contribute to these quality solutions. Figure 8.14 shows the mean distribution of operators within factorial function solutions for Traditional GP and the two mapping types in PAM DGP. (Standard is dropped for clarity since it also uses Huffman encoding, like Huffman-based PAM DGP, for function emphasis.)

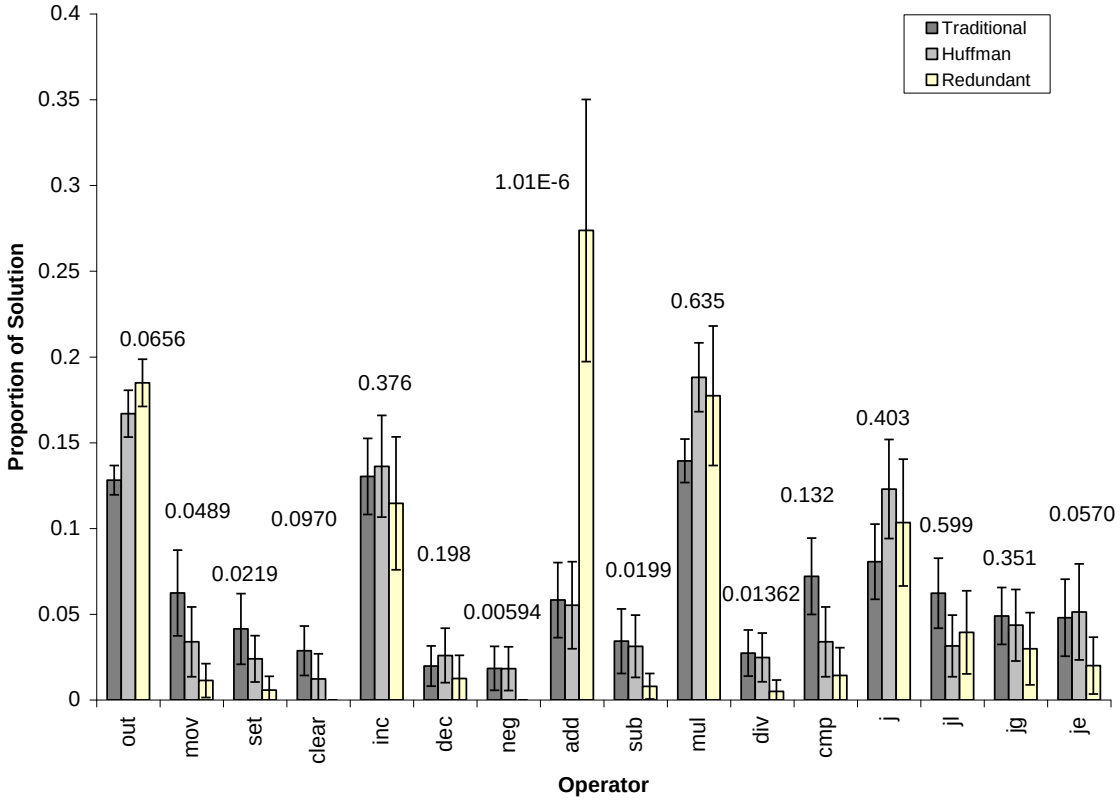


Figure 8.14. Mean operators as a proportion of total solutions for the factorial sequence when the success criterion is met or after 500 000 rounds over 50 trials. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. P-values corresponding to Huffman and Redundant mappings are displayed above each set of data points.

It is statistically significant at the 0.95 confidence interval that Redundant PAM DGP avoids move, set, subtract, and divide to a greater degree than all the other algorithms. All of those operators could be disruptive to the production of the factorial series which requires repeated multiplication and addition. Also significant at the 0.95 confidence interval is Redundant PAM DGP's emphasis on addition, and the top five operators given the most emphasis by Redundant PAM DGP are potentially useful in generating the factorial sequence (multiplication, addition, increment, jump, and output).

The number of operators used by each algorithm over 50 trials is given below (Figure 8.15).

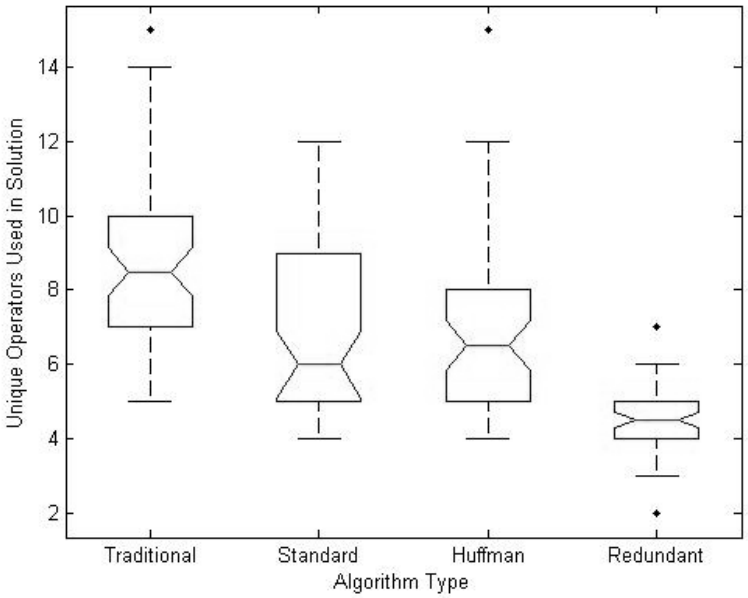


Figure 8.15. Boxplot indicating the number of operators constituting each solution for the factorial function when the success criterion is met or after 500 000 rounds over 50 trials. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times interquartile range.

Redundant PAM DGP uses a significantly lower number of operators per solution (leftmost boxplot), obviously statistically significant at the 0.95 confidence interval. The lowest number of operators used by Redundant PAM DGP was two (bottom outlier), producing a non-general (in particular, non-branching) attempt at a solution using repeated output and addition. Overall, Figures 8.14 and 8.15 indicate that Redundant PAM DGP generated its efficient general solutions through a reduction in function set size and appropriate emphasis of function set members.

The Fibonacci series represented a more difficult (second order) recursive function, and required only three operators in its natural recursive form (Equation 8.2): addition, output, and a jump operator. The Fibonacci series solutions' allocation of operators over 50 independent trials is shown below in Figure 8.16. Redundant PAM DGP placed a much higher level of emphasis on addition, output, and increment than the other algorithms (all very useful instructions for generating the Fibonacci series, and significant at the 0.95 confidence interval). The fourth most emphasized operator was the unconditional jump (with other jump variants close behind), allowing Redundant PAM DGP's top four operator choices to include the three required functions for the natural recursive solution of the Fibonacci series. The other algorithms failed to create the degree of preferential function emphasis exhibited by Redundant PAM DGP.

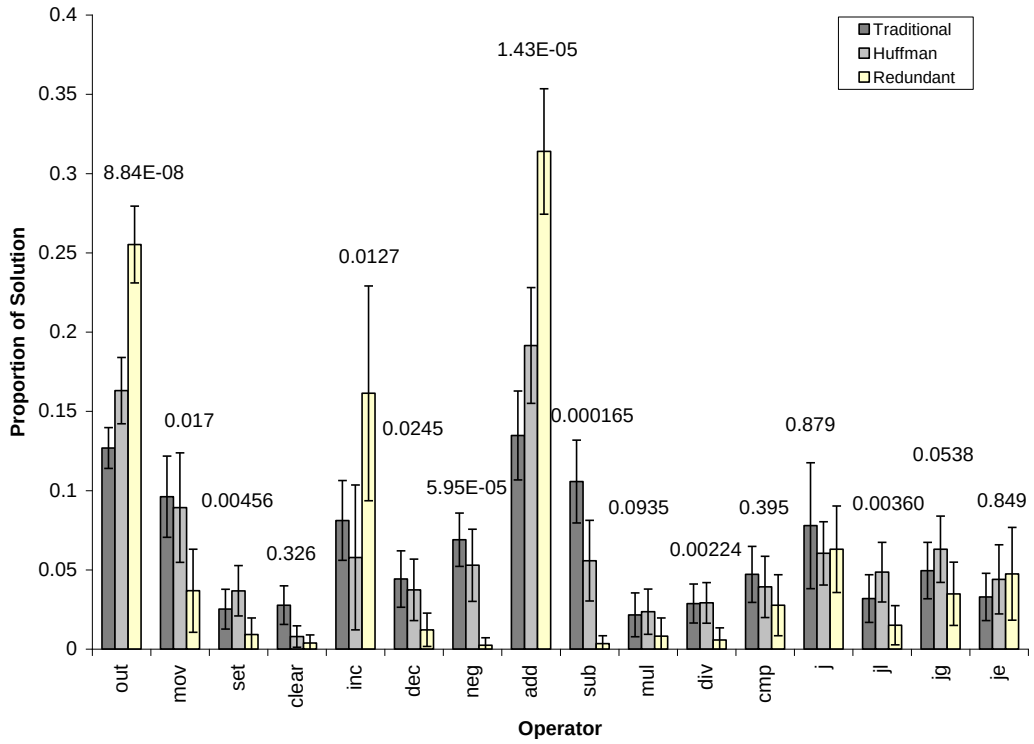


Figure 8.16. Mean operators as a proportion of total solutions for the Fibonacci series when the success criterion is met or after 500 000 rounds over 50 trials. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. P-values corresponding to Huffman and Redundant mappings are displayed above each set of data points.

The spread of data corresponding to raw number of operators used in each solution is given below in Figure 8.17. Redundant PAM DGP clearly finds solutions using a significantly more succinct function subset, with Traditional GP (fixed mapping) attempting to create solutions using a large number of the available functions. Redundant PAM DGP's ability to produce the largest number of solutions and the most efficient general solutions to the Fibonacci series (discussed in Section 8.3.3), indicates that it is creating better solutions through use of a smaller subset of the available functions than the other algorithms (Figure 8.17) and/or emphasis on appropriate operators in those sets (Figure 8.16).

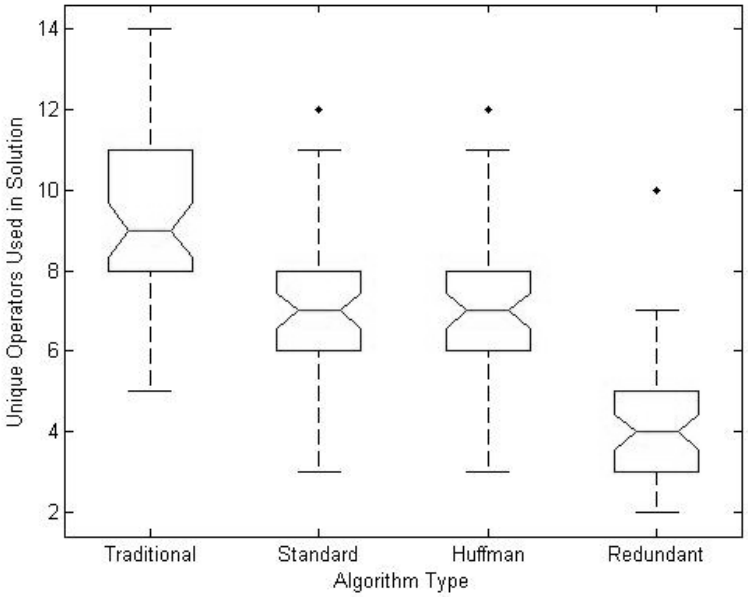


Figure 8.17. Boxplot indicating the number of operators constituting each solution for the Fibonacci series when the success criterion is met or after 500 000 rounds over 50 trials. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times interquartile range.

The third order Fibonacci series represents the highest order of recursion investigated in this work. As was the case for the regular (second order) Fibonacci series, the operators used in the natural recursive solution are a jump, addition, and output. The allocation of operators over 50 independent trials for the third order Fibonacci series is shown below in Figure 8.18.

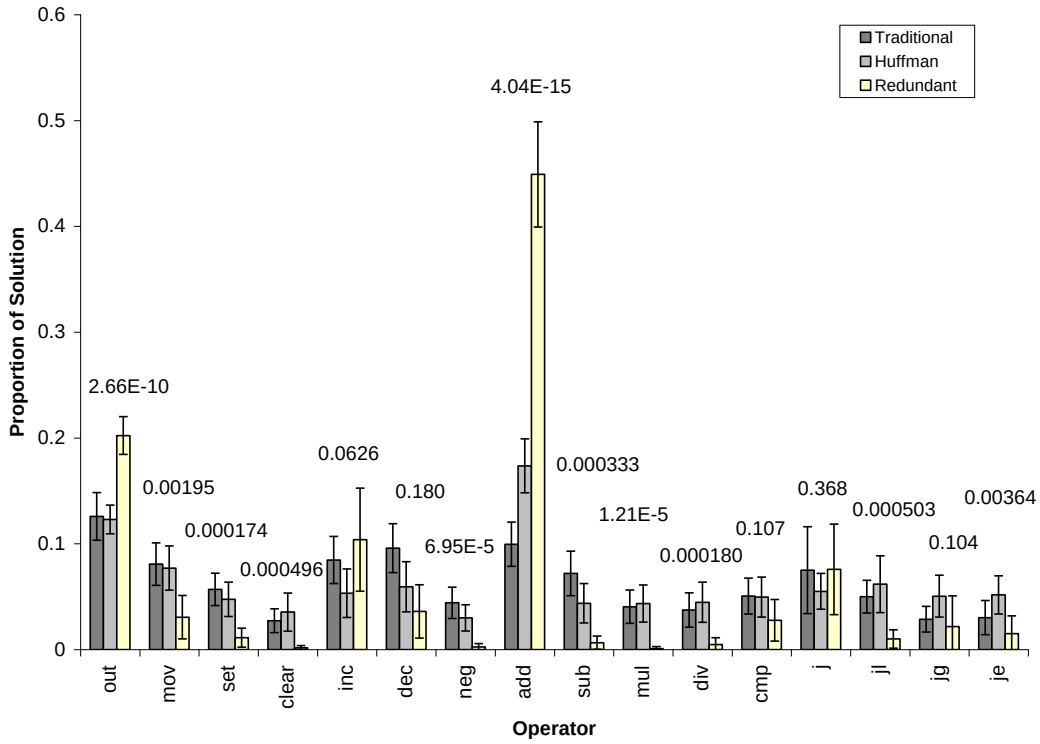


Figure 8.18. Mean operators as a proportion of total solutions for the third order Fibonacci series when the success criterion is met or after 500 000 rounds over 50 trials. Error bars reflect two-tailed t-distribution for the 0.95 confidence interval. P-values corresponding to Huffman and Redundant mappings are displayed above each set of data points.

Figure 8.18 clearly shows that Redundant PAM DGP correctly emphasizes the addition and output operators in its solutions to a much greater degree than Traditional GP and Huffman PAM DGP (significant at the 0.99 confidence interval). It also has a healthy emphasis of the unconditional jump function (as well as emphasizing increment, which can also be useful in solution construction and was actually incorporated in Redundant PAM DGP's general solution in Section 8.3.4). Traditional GP and Huffman PAM DGP have a comparatively even distribution of functions across their solutions. The raw number of unique function types used in solutions for every algorithm is given below in Figure 8.19.

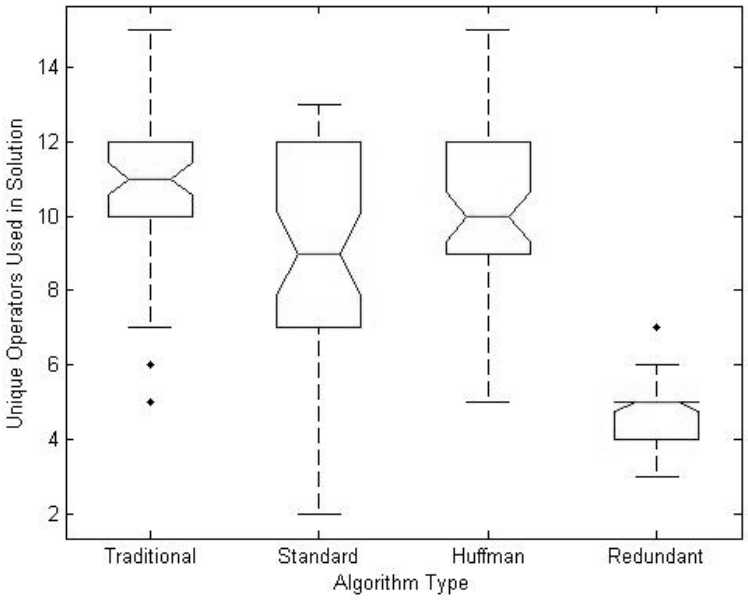


Figure 8.19. Boxplot indicating the number of operators constituting each solution for the third order Fibonacci series when the success criterion is met or after 500 000 rounds over 50 trials. Each box indicates the lower quartile, median, and upper quartile values. If the notches of two boxes do not overlap, the medians of the two groups differ at the 0.95 confidence interval. Points represent outliers to whiskers of 1.5 times the interquartile range.

Redundant PAM DGP once again clearly produces the most parsimonious solutions, using the fewest function types per solution. For this problem, where Redundant PAM DGP produced considerably more solutions than the other algorithms as shown in Section 8.3.4, the beneficial effect of the reduced function set size and appropriate function emphasis is the most salient.

8.5 Recursion Problems Summary

In this chapter, the abilities of Traditional GP, Standard Adaptive Mapping DGP, Huffman PAM DGP, and Redundant PAM DGP algorithms to automatically discover recursive solutions to progressively more difficult (higher order) recursive functions were investigated. Furthermore, these solutions were composed of a set of functions that did not include implicitly recursive operators defined *a priori*; thus the algorithms automatically learned to perform recursion. The factorial function was found to be easily solved by Traditional GP, Standard Adaptive Mapping, and Huffman PAM DGP; Redundant PAM DGP produced less overall solutions. All solutions for the factorial series were general, but Redundant PAM DGP was found to produce better quality (more efficient) general solutions capable of generating more of the factorial sequence within the maximum number of execution steps.

Redundant PAM DGP produced more solutions for the Fibonacci series, which is by definition second order recursive. Furthermore, it located those solutions in less tournament rounds than the other algorithms and produced more efficient general solutions capable of generating longer Fibonacci sequences within program termination constraints. Redundant encoding in PAM DGP was needed to solve the third order Fibonacci series problem with even partial repeatability: it found 14 solutions, while the only other algorithm to locate a (non-general) solution was Traditional GP. Redundant PAM DGP also found its solutions sooner than Traditional GP and created a novel mechanism for generating its sequence. The program content of the highest quality (capable of producing the longest sequence) solutions were examined for each function,

and the solutions produced by Redundant PAM DGP were entirely intron free. This indicates very efficient use of the function set to produce the best general solutions.

The number of unique functions used and the proportion of each solution allocated to each function were also examined. In each problem, Redundant PAM DGP was found to place more appropriate emphasis on relevant functions than Traditional GP or Huffman PAM DGP. For all problems Redundant PAM DGP also included less function types in its solutions. The collective analysis contained in this chapter demonstrated the ability of Redundant PAM DGP to automatically generate high quality recursive solutions, and indicated that these benefits were due to both trimming of the available function set and/or appropriate emphasis of the best available operators in the function set.

Chapter 9. Conclusion

9.1 Discussion: Parameterization and Limits of PAM DGP

The parameterization of PAM DGP typically changed between problems in this thesis, so some guidance and considerations for parameter selection in PAM DGP are in order as a basic guide to future application of the algorithm to new problems. The first problem investigated was the fairly trivial Maximum Output problem as posed by Margetts and Jones and used to introduce their Standard Adaptive Mapping algorithm [58]. From that analysis, the most obvious consideration is the size of the population used to solve a problem. PAM DGP used a population of 8 (4 mapping and 4 genotypes, the minimum required to conduct the steady state tournament of size 4) to solve the Maximum Output problem, with increased population size being detrimental to the solving of such a simple problem. On the other hand, for the harder Two Boxes regression problem, a population of 50 individuals (25 mappings and 25 genotypes) was required to produce a solution given the tournament round limits. There is a trade-off between population size in PAM DGP and the size of the probability table used for genotype-mapping combinations in the tournament, where a population of size P yields a grid of size $(P/2)^2$. In other words, there is a consideration required regarding the amount of initial genotype material so that there is a sufficient amount on which to perform search for harder problems, but not too much so that simpler problems are given an unreasonably large search space. The reasonable search space of 50 (25 genotypes, 25 mappings) was also found to perform well for medical classification benchmarks and

learning of first through third order recursive sequences. If the user is faced with an unknown problem, it is unknown whether the search space will be difficult or hard to traverse, and the user does not wish to experiment with preliminary runs, a recommendation would be to use a larger search space: if the problem requires the larger amount of initial genotype material to explore, it will be present; if the problem is more trivial, it may not be solved as quickly, but it will likely be solved.

Genotype and mapping mutation and crossover rates and types are a design decision that warrants some discussion as well, as PAM DGP is a flexible algorithm that allows different mutation and crossover rates, and even types, for either genotype or mapping. In the MAX problem, a fairly straightforward approach was taken to benchmark against Margetts and Jones, namely, equivalent mutation and crossover rate were used for both mappings and genotypes. In particular, point mutation was used at a rate of 0.1 and crossover of equal-sized segments was performed at a rate of 0.9. Point mutation in the mapping population for the countingOnes function (Equation 3.1) allowed smaller changes in frequency nicely by changing individual bits as opposed to applying an XOR mask to an entire frequency segment. The high rate of crossover in the mapping population allowed expedient exploration of the existing search space. This somewhat naïve parameterization worked well enough for the trivial MAX regression domain.

The genetic operator parameterizations were then changed from the MAX problem to better suit the considerably harder Hénon mapping and Two Boxes regression problems, with the salient differences being a separation of genotype/mapping crossover rates and mutation type and rates between the two populations. Increased mutation and

crossover rates (0.5 and 0.9, respectively) were used in the genotypes to allow exploration against the backdrop of more persistent (due to lower genetic operator rates) mappings. Crossover type was the same for both populations, but mutation type differed between genotype and mapping individuals: Point mutation was still used in the mappings to ensure a number of frequencies per function set member were incrementally explored for both Huffman and Redundant (rather than the radical change afforded by an XOR mask); whereas an instruction-level XOR mask mutation operator (the instruction chosen with uniform probability) was used for the genotype mutation operator to enhance the exploration of the genotype against the more persistent mapping contexts. Identical rates and types, with the same rationale, were used for genotypes and mappings in the medical classification benchmarks.

Learning recursive functions, however, involved raising the mutation and crossover rates from 0.1 each to 0.5 and 0.9, respectively, for the mapping population. In this case the more stable context in the mapping population against which the genotypes could evolve was traded for greater exploration of mapping alternatives. As demonstrated in Chapter 8, this was rewarded with tuned function sets that reflected the function set members required for the natural recursive functions for the sequences provided. In the harder regression and medical classification problems, however, higher rates such as these were found to provide too unstable a mapping environment for effective genotype search in preliminary experiments. If it is suspected that a problem requires, or the user simply desires, a function set that will be highly tuned and/or reduced in size, higher genetic operator rates would be appropriate for the mapping

population. This is another element of the flexibility and power of the PAM DGP algorithm, but it does mean another design choice for the user.

PAM DGP also introduces two parameterizations that are unique to this algorithm's probability table and thus affect selection and search: learning rate (α) and noise (γ). To review, the learning rate dictates how much emphasis is placed on current fitness values as opposed to previous search, and noise is introduced to the probability table columns representing genotype-associations if the noise threshold is exceeded by any row (mapping) of the column in order to prevent premature convergence. The learning rate was set at a conservative rate of 0.1 for all algorithms, where higher rates could disrupt effective search using the table. The noise threshold was set at 0.95 for all problems except for learning or recursive sequences, where it was set at 0.8. Generally, lower settings for the noise threshold would result in a more stochastic search (i.e., exploration), but this is to be balanced against concerns about wasting search time around local optima. A lower noise threshold more readily allowed avoidance of local optima at the risk of not exploring space around those optima, which proved more effective in preliminary trials of the recursion problem and thus resulted in the choice of lowering the rate a small degree. The parameters of PAM DGP, it should be noted, are fairly robust and not sensitive to small changes. That is, users need not be concerned that small changes (at the second decimal place, for instance) in the parameters will perturb the quality of solutions greatly, and users can expect gradual fitness change with incremental parameter changes.

Aside from design choices involving specific parameterizations of PAM DGP, the algorithm does have its limitations—after all, no single algorithm is best for all problems

according to the No Free Lunch theorem. As mentioned earlier, there is additional overhead associated with developmental systems that evolve mappings because these systems must discover both the appropriate genotype solution and the best mapping for a function set to match, whereas traditional GP need only focus on locating the appropriate genotype. It is thus more difficult problems, often complex function sets or those involving potentially extraneous symbols, which are best suited to evolved mapping algorithms. For such problems, PAM DGP is able to outperform Traditional GP by discovering relevant subsets of symbols and concentrating on forming solutions using appropriate emphasis of symbols and selection of the relevant subset of the function set, decreasing the size of the DGP's search space. Regression problems, where a number of alternative combinations of choices from the function set can provide sufficient approximations to the fitness cases, may not be most *efficiently* handled using PAM DGP. However, this is simply due to increased overhead of mapping selection. PAM DGP thus may take more time to find the solution than, say, Traditional GP, but the function set will be appropriately large (or small). The flipside of this point is that the ability of PAM DGP, especially using the adaptive redundant mapping, to tailor the function set (and thus the search space) to a problem will allow it to more quickly solve difficult and more complex problems than traditional alternatives and leave the task of determining an appropriate function set to the algorithm rather than assuming that all members of a function set are equally relevant in a problem space which is not fully understood. We have shown that the adaptive redundant mapping in PAM DGP is a far more effective mapping choice than Huffman-encoded mapping for trimming, and emphasizing appropriate members of, function sets in such interesting problem domains.

Aside from choosing when to favor PAM DGP and developmental systems over Traditional GP approaches, there is also the consideration of when to use PAM DGP over other machine learning (ML) techniques in general. All ML approaches can be seen to address three basic design principles: cost function (goals/objectives), representation, and credit assignment policy for search. Design decisions based on these issues determine the quality and transparency of solutions, the computational efficiency of the ML paradigm, and the applicability of the technique to a problem domain.

All machine learning approaches involve a metric to express the suitability of the current solution. In the example of training a classifier as in Chapter 7, a distance metric (error measure) is evaluated over a set of training exemplars. Depending on the ML model, there may be implicit constraints dictating the design of the distance metric. As an example, the smoothness constraint of neural networks and kernel methods implies that the relation between the objective function and the model's specifiable parameters be differentiable [34, 62]. In contrast, evolutionary computation representations have no such constraint on the formulation of the goals and representation. The price for this flexibility in EC is that the method for credit assignment is not as direct, and exploration of the search space is emphasized over exploitation during search. The inherent flexibility of EC paradigms also means that there is a natural mechanism for incorporating multiple objectives. This allows a way of minimizing both false positive and detection rates in a classification domain, which is of particular interest when the data is unbalanced.

Machine learning paradigms incorporate their own specific models for representing a solution. For instance, neural networks use a connected topology of

neurons, and kernel methods rely on the "kernel trick," where a non-linear mapping of the original input space to a very high dimensional space is used, after which linear separability allows recombination into a final solution [34]. Constraints in the representation of a solution from the ML model are a source of both strengths and limitations that ought to be considered when applying the model to a problem domain. Neural networks and kernel methods explicitly use all input features rather than search for the best subset of applicable features, giving an efficient mechanism for credit assignment. However, these models are not particularly transparent and involve limiting properties, such as not providing support for learning temporal relationships. EC methods such as GP, however, are applicable to both non-temporal and temporal problem domains (such as the recursion problems of Chapter 8), and are biased against incorporating the entire feature set of a problem within a solution (unlike neural nets and kernel methods). Also, EC methods allow the flexibility of adopting a discrimination-based or novelty detection (appropriate for single class learners) policy when building classifiers. In contrast, decision trees will always use a discrimination-based classification policy [62]. EC models, in summary, provide flexibility in representation leading to the discovery of novel and transparent solutions, but this may come at the expense of computational overhead.

The "credit assignment" problem is the process by which a ML paradigm adapts the free parameters associated with the representation (such as weights in neural networks and kernel methods, and tests in a decision tree). Means of effectively adapting parameters in neural networks are based on classical numerical optimization algorithms such as Quasi-Newtonian, quadratic programming, and Lagrangian methods [34], and

decision trees use entropy-based partitioning algorithms [62]. Both means of credit assignment use an exploitation or greedy policy, so they are particularly effective in classification and function approximation domains. Problem domains where there is a considerable delay between actions proposed by the model and pay off from the problem environment (such as control problems or environments with incomplete information such as network routing and path planning) can benefit from a more exploratory approach to credit assignment. Such exploratory approaches mean that model parameterizations may be accepted that perform worse than the current model. Machine learning models that are biased toward exploratory approaches to credit assignment include Evolutionary Computation, simulated annealing, and Boltzman machines. It is also possible to address credit assignment using hybrid models, such as Genetic Algorithms with numerical optimization or evolving neural networks with Evolutionary Computation (which incorporates design of both neural network connectivity and weight value optimization).

As a general procedure to guide the choice of ML paradigm for an application domain, a first step would be to apply a simple linear model trained using a greedy credit assignment mechanism to establish a performance baseline. If performance is not found to be satisfactory, the practitioner ought to move to a non-linear ML model. This procedure will permit advantages in terms of determining computational efficiency, transparency, and qualification of properties inherent in an unknown problem domain prior to the application of increasingly complex ML models that are more expensive to both build and analyze.

9.2 Summary and Conclusions

This work presented a new developmental GP algorithm that models the symbiotic coevolution of the genetic code (biological codon to amino acid) and genotype in nature. Previous similar developmental systems were discussed, as were pathologies of cooperative coevolution that require addressing in order to ensure the design of an efficient search algorithm. Shortcomings of the most relevant previous coevolutionary algorithm, the Standard Adaptive Mapping algorithm of Margetts and Jones [57-59], were exposed and illustrated empirically. In particular the previous Adaptive Mapping algorithm was demonstrated to suffer from a lack of exploration of the search space and the Red Queen Effect, with associated repeated loss of context, fitness spiking, and an overall lack of fitness-based performance.

The PAM DGP algorithm was then introduced, initially operating on populations of genotypes and mappings using the same structure and encoding process as the Adaptive Mapping algorithm. The PAM DGP algorithm components responsible for overcoming the drawbacks of the Adaptive Mapping algorithm, and the Red Queen Effect in general, were combined elitism and a novel probability table that allowed dynamic fitness proportionate selection of promising genotype-mapping pairs. The PAM DGP algorithm was shown to outperform the Adaptive Mapping algorithm on the simple Maximum Output benchmark used to introduce the latter algorithm to the literature [58], and the Two Boxes and Hénon map regression problems. Having established empirically that the algorithm behind PAM DGP provided considerable design benefits, a population of novel (more developmentally sound) adaptive redundant mappings were incorporated into the PAM DGP framework to replace the Huffman-encoded mappings. While PAM

DGP with the adaptive redundant mappings provided too much search overhead to be effective at simple regression problems, it was found to perform better than Huffman encoding on harder regression problems and yield highly effective classifiers for the more difficult benchmark medical classification problems, in both cases by tailoring the available function set and emphasizing appropriate members. In effect, by concentrating on explicitly identifying the most suitable symbols from the function set, the PAM DGP algorithm (with redundant mapping) was able to provide an efficient mechanism for reducing the size of the search space relative to Traditional GP, Standard Adaptive Mapping DGP, and PAM DGP using Huffman encoding.

PAM DGP also produces the most efficient general solutions over the factorial, Fibonacci, and third order Fibonacci recursive functions compared to the competing algorithms just listed. That is, it produced the solutions that were best able to generate the longest correct function sequence up to the allowed execution limit. Furthermore, its best (most general) solutions for each problem were shown to be entirely intron-free. Given higher order recursion problems (2nd and 3rd order Fibonacci), PAM DGP also generated the largest number of solutions and did so in less tournament rounds than any algorithm it was tested against. Redundant PAM DGP was also shown to evolve its genetic code mappings so as to emphasize the operators useful for the natural general recursive solutions for each function's sequence.

9.3 Future Work

PAM DGP is a developmental system that models the coevolution of genetic code and genotype, as well as the redundant nature of the biological genetic code. In so doing,

PAM DGP provides an effective means of dealing with the Red Queen Effect in two population cooperative coevolution and yields solutions to non-trivial regression, medical classification benchmarks, and learning of recursive sequences using generic machine language-based function sets by emphasizing appropriate members of the function set and reducing it to a problem-tailored subset. There are a couple of directions that present themselves as an application of this algorithm in future work. The first would be to utilize the algorithm to fit a function set to a difficult problem type where the best function set choices are not known, and then allow Traditional GP to use the finished function set to solve similar problems in the future (provided the problem is suitable and static). The algorithm could be used to determine appropriate function sets for problem domains, in essence. The second natural extension of the work would be to incorporate the ability to efficiently evolve appropriate genetic codes into a larger developmental framework. After all, in biology, our gene translation process that has evolved over time is only a small component of the developmental processes in our cells, let alone our entire body. There is currently a strong movement toward shifting the focus of traditional evolutionary algorithms to the construction of algorithms that have more of a basis on our current understanding of molecular and evolutionary biology. In other words, there appears to be a shift underway from engineering-based "artificial evolution" to biologically inspired "computational evolution" systems that incorporate more advanced biological notions such as self-modification and feedback that are not implemented in most current algorithms [8]. It is hoped that this work on properly and efficiently evolving the genetic code may contribute to more extensive developmental systems in the future.

In addition to using the algorithm in larger frameworks, there are a number of opportunities for future work in refining and building on the PAM DGP algorithm itself. Extensions to the algorithm with the goal of improving the general usability are a possibility: while the algorithm is robust and flexible, this comes at a cost of having the user make a number of parameterization choices. Future implementations could examine the potential for automatic and adaptive parameterization of variables such as mutation and crossover rates for mapping and genotypes populations while maintaining reciprocal contexts for efficient evolution of both types of individuals. Another possibility for increased automation of parameterization is dynamically setting PAM DGP's learning rate and noise threshold throughout the execution of the algorithm for particular problems.

There is also the potential for additional investigation of the theory behind the performance benefits of PAM DGP that were discovered in this thesis. The fitness-based and efficiency metrics were shown to be associated with both a reduction in size of the function set and emphasis of particular function set members. To study on the effect of the interpretation of instructions from the genotype, and the effect of the complexity of instructions or uniformity in instruction interpretation on problem difficulty, would be enlightening to determine how to best set up the problem space for efficient PAM DGP search. That is, the ability of PAM DGP to trim the function set and explicitly emphasize function set members may be affected not only by function set size, but by the nature of the instructions themselves, where the uniformity with which they are interpreted contributes to more efficient mapping search.

Another theoretic issue is that while it is evident that adaptive redundant mappings are effectively reducing search space through their evolution of the function set, it is not known whether they introduce solution programs of increased sizes in some cases while using the reduced function sets to more effectively solve problems. Individuals in these experiments were fixed in terms of the length of their binary strings, but if variable sized individuals were attempted in the future, it would be of interest to investigate the length of the solutions using the reduced function set compared to static, globally defined mapping solutions. For the fixed string genotypes used here, it would very likely be the case that Huffman encoded mappings would produce longer solution lengths simply by virtue of the Huffman algorithm being a compression algorithm—it will attempt to make the binary genotype include as many instructions as it possibly can with variable sized encodings. In contrast, our adaptive redundant mapping uses fixed size encodings, as would Traditional GP. Thus, it is expected that the adaptive redundant mapping-based individuals would produce solution lengths on par with Traditional GP.

It is very promising that PAM DGP with redundant encodings was found to produce semantically high quality (meaning efficient and succinct) recursive solutions given a generic machine language function set. These studies relied on a preset limit of execution steps to terminate recursive loops. In fact, this limit was indirectly used as a means of roughly measuring semantics: semantically better solutions generated more correct values prior to forced termination (and were even found to be intron-free). Thus, in our search for a semantically good solution from a generic function set, we turned concern for termination on its head. Future work could continue to involve measuring semantic goodness in the same way in a first stage of an algorithm, followed by providing

the semantically best solution's loop contents to a higher order function (as described by Yu in [101, 104]) with ensured termination. PAM DGP has proved capable of learning recursive functions and emphasizing appropriate function set members, so future work will naturally investigate the use of the adaptive redundant mappings in PAM DGP to solve additional, more difficult, optimization tasks from real world domains with function sets consisting of higher level mathematical operators (such as square root, log, *et cetera*). If successful, such a system would finally provide semantically high quality recursive solutions capable of self-termination—all while allowing the recursion process to definitively be evolved and not introduced as a recursion-enabling operator within the function set.

In terms of future work regarding applying PAM DGP to difficult recursive problems, readers may have noted that the Hénon mapping was posed as a regression problem rather than a recursive sequence, even though the equation is recursive in nature. As mentioned in Chapter 4, the Hénon mapping was evaluated by Margetts [57] in regression form, so we wished to compare performance. Also, the Hénon mapping was found to be a very challenging problem when posed as a recursive sequence. As a "chaotic attractor," by its nature, the values it produces in sequence appear to have little or no underlying pattern—it is only when plotted in two or three dimensions that an overarching pattern can be observed. An algorithm learning the sequence would thus require a considerably longer prefix of the sequence than the 10 used by Huelsbergen or this work. Even given a considerably longer prefix, the function still appears to generate a mostly random sequence. Difficulty in error calculation is compounded by the fact that the sequence is a set of real-valued numbers rather than an integer sequence. Finally, the

machine language function set does not possess all the higher level operators used in the Hénon equation (Equation 5.2), so the algorithm must figure out not only the solution to the Hénon equation, but how to form the required constants and square operator used within it. A starting point would be to handle the Hénon equation with a function set including higher level operators, and a significant sequence prefix with which to learn the function (perhaps 500 values instead of 10). The recursive solution to the Hénon mapping from a general function set poses a very interesting and challenging real world problem.

Another real world application that could be attempted involves application of PAM DGP to network intrusion detection systems (IDSs). Some of my research colleagues have produced IDS systems using linear GP (LGP), showing LGP to be a promising approach to classification for IDS. The work of Song, Heywood, and Zincir-Heywood using GP IDS for wired networks [88] has used TCP dump data provided by DARPA, or the KDD-99 data set, which consists of a large collection of connection statistics. The work on wireless systems in LaRoche and Zincir-Heywood [54] was generated by the researchers. In the work on wired networks [88], the learning problem was to create a detector (a classifier) that could distinguish between good (normal) and bad (intrusion or attack) connections. Each connection was described by 41 features, 9 basic features and 32 derived features. Work on wireless networks [54] used management frame packets for the IEEE 802.11 standard, with 7 features used to label a connection as an attack (de-authentication attack only) or a normal connection. The choice of the features from the packet in both cases was made based on *a priori* knowledge of the attack type.

In a PAM DGP-based IDS, features from the packets could be incorporated as members of the function set, with the adaptive mapping being able to choose relevant packet features via trimming of the function set. The system would thus determine relevant packet features, where the user has to typically do this *a priori*. The set of available operations for the packet features would remain constant in this application scenario. The system would automatically determine connection features indicative of an attack without any decisions by the user. With the ability of PAM GP to handle large function sets and emphasize certain elements in those sets to provide tailored solutions, the IDS system may even have the potential to be expanded to not only simply classify a connection as normal or an attack, but refine the diagnosis to specify the *type* of attack.

In a related development, Kayacik, Heywood, and Zincir-Heywood [44] use a GP system to create an attack of system calls that cannot be recognized by a targeted IDS because it appears sufficiently similar to normal behavior. By generating better attacks, the design of detectors can be improved. The implementation uses only anomaly rate feedback from the detector, where other approaches have assumed additional knowledge of privileged detector information. In the implementation, the function set of instructions used to construct the attack is based on a particular use case. In [44], the top 15 system calls of the *Traceroute* application are chosen as the set with which to build the attack. By incorporating PAM DGP and evolving a function set of system calls in the IDS, it would be possible to tune the function set to the detector during the evolutionary process. This would automatically provide the appropriate system calls for attack of the use case on which the detector is built without the necessity of attempting to provide the most common system calls of the exploited application in advance.

This chapter presents a number of ideas for theoretical and practical application of PAM DGP and adaptive redundant mappings in possible future work. The author has endeavored to produce a novel developmental genetic programming algorithm capable of overcoming cooperative coevolutionary pathologies, and an associated adaptive redundant mapping capable of handling complex function sets. It is hoped that the robustness and flexibility of PAM DGP and its associated redundant mapping, as well as the lessons learned in this thesis based on its construction and application, will provide a useful contribution to future work in the area of modeling and efficiently evolving the genetic code in developmental systems.

References

1. Altenberg, L., Evolving Better Representations through Selective Genome Growth. In *Proceedings of the IEEE World Congress on Computational Intelligence*, (Orlando, Florida, 1994), IEEE Press, 182-187.

2. Altenberg, L. Genome Growth and the Evolution of the Genotype-Phenotype Map. In Banzhaf, W. and Eeckman, F. eds. *Evolution and Biocomputation: Computational Models of Evolution*, Springer-Verlag, Berlin, 1993, 205-259.

3. Angeline, P. and Pollack, J., Coevolving high-level representations. In *Proceedings of Artificial Life III*, (Santa Fe, California, 1994), Addison-Wesley, 55-71.

4. Banzhaf, W. Artificial Regulatory Networks and Genetic Programming. In Riolo, R. and Worzel, B. eds. *Genetic Programming in Theory and Practice*, Kluwer Academic, Norwell, 2003, 43-61.

5. Banzhaf, W., Genotype-Phenotype Mapping and Neutral Variation. In *Parallel Problem Solving from Nature III*, (Jerusalem, Israel, 1994), Springer-Verlag: Berlin, 322-332.

6. Banzhaf, W. On Evolutionary Design, Embodiment and Artificial Regulatory Networks. In Iida, F.e.a. ed. *Embodied Artificial Intelligence*, Berlin, Springer-Verlag, 2004, 284-292.

7. Banzhaf, W., On the Dynamics of an Artificial Regulatory Network. In *Advances in Artificial Life, Proceedings of the 7th European Conference (ECAL-2003)*, (Dortmund, Germany, 2003), Springer-Verlag, 217-227.

8. Banzhaf, W., Beslon, G., Christensen, S., Foster, J., Kepes, F., Lefort, V., Miller, J., Radman, M., and Ramsden, J. From artificial evolution to computation evolution: a research agenda. *Nature Reviews Genetics*, *7* (9): 729-735, 2006.

9. Bean, J. Genetic Algorihtms and Random Keys for Sequencing and Optimization. *ORSA Journal of Computing*, *6* (2): 154-160, 1994.

10. Bentley, P., Adaptive Fractal Gene Regulatory Networks for Robot Control. In *Workshop on Regeneration and Learning in Developmental Systems in the Genetic and Evolutionary Computation Conference (GECCO 2004)*, (Seattle, Washington, 2004).

11. Bentley, P. Investigations into Graceful Degradation of Evolutionary Developmental Software. *Journal of Natural Computing*, *4*: 417-437, 2005.

12. Bentley, P. and Kumar, S., Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, (Orlando, Florida, 1999), Morgan Kaufmann, 35-43.

13. Bongard, J., Evolving Modular Genetic Regulatory Networks. In *Proceedings of the IEEE 2002 Congress on Evolutionary Computation (CEC 2002)*, (Honolulu, Hawaii, 2002), IEEE Press, 1872-1877.

14. Bongard, J. and Pfeifer, R. Evolving Complete Agents using Artificial Ontogeny. In Hara, F. and Pfeifer, R. eds. *Morpho-functional Machines: The New Species*, Springer-Verlag, 2003, 237-258.

15. Brameier, M. and Banzhaf, B. *Linear Genetic Programming*. Springer, Berlin, 2007.

16. Brameier, M. and Banzhaf, W. A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining. *IEEE Transactions on Evolutionary Computation*, *5* (1): 17-26, 2001.

17. Brave, S. Evolving recursive programs for tree search. In Angeline, P. and Kinnear, K., Jr. eds. *Advances in Genetic Programming 2*, MIT Press, Cambridge, MA, 1996, 203-219.

18. Bucci, A. and Pollack, J., On Identifying Global Optima in Cooperative Coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2005)*, (Washington, DC, USA, 2005), ACM Press: New York, 539-544.

19. Bui, L., Abbass, H. and Essam, D. Coevolution of genotype-phenotype mapping to solve highly epistatic problems (TR-ALR-200506010) *The Artificial Life and Adaptive Robotics Laboratory: ALAR Technical Report Series*, University of New South Wales, Canberra, Australia, 2005.

20. Cliff, D. and Miller, G., Tracking the Red Queen: Measurements of adaptive progress in co-evolutionary simulations. In *Advances in Artificial Life: Proceedings of the Third European Conference on Artificial Life (ECAL 95)*, (Granada, Spain, 1995), Springer-Verlag: London, 200-218.

21. Crick, F. The origin of the genetic code. *Journal of Molecular Biology*, *38* (3): 367-379, 1968.

22. de Jong, E. and Pollack, J. Ideal Evaluation from Coevolution. *Evolutionary Computation*, *12* (2): 159-192, 2004.

23. Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S. and Froelicher, V. International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, *64*: 304-310, 1989.

24. Di Caro, G. and Dorigo, M. AntNet: Distributed Stigmergetic Control for Communication Networks. *Journal of Artificial Intelligence Research*, *9*: 317-365, 1998.

25. Ebner, M., Langguth, P., Albert, J., Shackleton, M. and Shipman, R., On Neutral Networks and Evolvability. In *Proceedings of the 2001 Congress on Evolutionary Computation (CEC 2001)*, (Seoul, South Korea, 2001), IEEE Press, 1-8.

26. Ebner, M., Shackleton, M. and Shipman, R. How Neutral Networks Influence Evolvability. *Complexity*, *7* (2): 19-33, 2001.

27. Ficici, S. and Pollack, J., Challenges in Coevolutionary Learning: Arms-Race Dynamics, Open-Endedness, and Mediocre Stable States. In *Sixth International Conference on Artificial Life*, (Los Angeles, California, 1998), MIT Press, 238-247.

28. Freeland, S. The Darwinian Genetic Code: An Adaptation for Adapting? *Genetic Programming and Evolvable Machines*, *3* (2): 113-127, 2002.

29. Gathercole, C. and Ross, P., An Adverse Interaction between Crossover and Restricted Tree Depth in Genetic Programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, (Stanford, California, 1996), MIT Press: Cambridge, MA, 291-296.

30. Gleick, J. *Chaos: Making a New Science*. Penguin Books, New York, 1987.

31. Goldberg, D., Korb, B. and Deb, K. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, *3* (5): 493-530, 1989.

32. Gordon, T. and Bentley, P., Development Brings Scalability to Hardware Evolution. In *Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*, (Washington, DC, USA, 2005), IEEE Computer Society, 272-279.

33. Handley, S., A new class of function sets for solving sequence problems. In *Genetic Programming 1996: Proceedings of teh First Annual Conference*, (Stanford, California, 1996), MIT Press, 301-308.

34. Herbrich, R. *Learning Kernel Classifiers: Theory and Algorithms*. MIT Press, Cambridge, 2002.

35. Hornby, G., Lipson, H. and Pollack, J. Generative Representations for the Automated Design of Modular Physical Robots. *IEEE Transactions on Robotics and Automation*, *19* (4): 703-719, 2003.

36. Hornby, G., Takamura, S., Yamamoto, T., Fujita, M. Autonomous Evolution of Dynamic Gaits with Two Quadruped Robots. *IEEE Transactions on Robotics*, *21* (3): 402-410, 2005.

37. Huelsbergen, L., Abstract Program Evaluation and its Application to Sorter Evolution. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC 2000)*, (San Diego, California, 2000), IEEE Press, 1407-1414.

38. Huelsbergen, L., Fast Evolution of Custom Machine Representations. In *Proceedings of the 2005 Congress on Evolutionary Computation (CEC 2005)*, (Edinburgh, UK, 2005), IEEE Press, 97-104.

39. Huelsbergen, L., Finding General Solutions to the Parity Problem by Evolving Machine-Language Representations. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, (Madison, Wisconsin, 1998), Morgan Kaufman, 158-166.

40. Huelsbergen, L., Learning Recursive Sequences via Evolution of Machine-Language Programs. In *Genetic Programming 1997: Proceedings of the Second International Conference*, (Stanford, California, 1997), Morgan Kaufman, 186-194.

41. Huelsbergen, L., Toward Simulated Evolution of Machine-Language Iteration. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, (Stanford, CA, 1996), MIT Press, 315-320.

42. Kargupta, H. A striking property of genetic code-like transformations. *Complex Systems*, *13*: 1-32, 2001.

43. Kargupta, H. and Ghosh, S. Toward Machine Learning Through Genetic Code-like Transformations. *Genetic Programming and Evolvable Machines*, *3*: 231-258, 2002.

44. Kayacik, H., Heywood, M. and Zincir-Heywood, A., Evolving Buffer Overflow Attacks with Detector Feedback. In *Applications of Evolutionary Computing: Proceedings of EvoWorkshops 2007*, (Valencia, Spain, 2007), Springer, (upcoming).

45. Keller, R. and Banzhaf, W., The Evolution of Genetic Code in Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, (Orlando, Florida, 1999), Morgan Kaufman: San Francisco, 1077-1082.

46. Keller, R. and Banzhaf, W., Evolution of Genetic Code on a Hard Problem. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, (San Francisco, California, 2001), Morgan Kaufman: San Francisco, 50-56.

47. Keller, R. and Banzhaf, W., Genetic Programming using Genotype-Phenotype Mapping from Linear Genomes in Linear Phenotypes. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, (Stanford, California, 1996), MIT Press: Cambridge, MA, 116-122.

48. Kimura, M. Evolutionary rate at the molecular level. *Nature, 217*: 624-626, 1968.

49. Koza, J. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, Cambridge, MA, 1994.

50. Koza, J. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic, Norwell, Massachusetts, 2003.

51. Koza, J. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992.

52. Koza, J., Streeter, M. and Keane, M. Automated Synthesis by Means of Genetic Programming of Complex Structures Incorporating Reuse, Parameterized Reuse, Hierarchies, and Development. In Riolo, R. and Worzel, B. eds. *Genetic Programming Theory and Practice*, Kluwer Academic Norwell, 2003, 222-237.

53. Langdon, W. and Poli, R., An Analysis of the MAX Problem in Genetic Programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, (Stanford, California, 1997), Morgan Kaufman, 222-230.

54. LaRoche, P. and Zincir-Heywood, A., 802.11 De-authentication Attack Detection using Genetic Programming. In *Proceedings of the 9th European Conference on Genetic Programming (EuroGP 2006)*, (Budapest, Hungary, 2006), Springer, 1-12.

55. Lodish, H., Baltimore, D., Berk, A., Zipursky, S., Matsudaira, P. and Darnell, J. *Molecular Cell Biology*. Scientific American Books, New York, NY, 1995.

56. Lohn, J., Linden, D., Hornby, G., Kraus, W., Rodriguez, A., and Seufert, S. Evolutionary Design of an X-Band Antenna for NASA's Space Technology 5 Mission. *Proceedings of the 2004 IEEE Antenna and Propagation Society International Symposium and USNC/URSI National Radio Science Meeting, 3*: 2313-2316, 2004.

57. Margetts, S. *Adaptive Genotype to Phenotype Mappings for Evolutionary Algorithms*. Ph.D. Thesis, Cardiff University, Wales, Great Britain, 2001.

58. Margetts, S. and Jones, A., An Adaptive Mapping for Developmental Genetic Programming. In *Proceedings of the Fourth European Conference on Genetic Programming (EuroGP 2001)*, (Lake Como, Italy, 2001), Springer Verlag: Berlin, 97-107.

59. Margetts, S. and Jones, A., Phlegmatic Mappings for Functional Optimisation with Genetic Programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2000)*, (Las Vegas, Nevada, 2000), Morgan Kaufman, 82-89.

60. Miller, J., Evolving a Self-Repairing, Self-Regulating, French Flag Organism. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, (Seattle, Washington, 2004), Springer-Verlag, 129-139.

61. Miller, J. and Banzhaf, W. Evolving the Program for a Cell: From French Flags to Boolean Circuits. In Bentley, P. and Kumar, S. eds. *On Growth, Form, and Computers*, Academic Press, New York, New York, 2003, 278-302.

62. Mitchell, T. *Machine Learning*. McGraw-Hill, Singapore, 1997.

63. Moriarty, D. and Miikkulainen, R. Efficient Reinforcement Learning through Symbolic Evolution. *Machine Learning*, *22*: 11-32, 1996.

64. Morrison, J. and Oppacher, F., A General Model of Co-evolution for Genetic Algorithms. In *Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA '99)*, (Portoroz, Slovenia, 1999), Springer-Verlag, 262-268.

65. Murao, H., Tamaki, H. and Kitamura, S., A Coevolutionary Approach to Adapt the Genotype-Phenotype Map in Genetic Algorithms. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC '02)*, (Honolulu, HI, 2002), IEEE Press, 1612-1617.

66. Newman, D., Hettich, S., Blake, C. and Merz, C. UCI Repository of machine learning databases [http://www.ics.uci.edu/~mlearn/MLRepository.html], University of California, Department of Information and Computer Science, Irvine, CA, 1998.

67. O'Neill, M. and Brabazon, A., mGGA: The meta-Grammar Genetic Algorithm. In *Proceedings of the 8th European Conference on Genetic Programming (EuroGP 2005)*, (Lausanne, Switzerland, 2005), Springer: Berlin, 311-320.

68. O'Neill, M. and Ryan, C. Grammatical Evolution. *IEEE Transactions on Evolutionary Computation*, *5* (4): 349-358, 2001.

69. O'Neill, M. and Ryan, C., Grammatical Evolution by Grammatical Evolution: The Evolution of Grammar and Genetic Code. In *Proceedings of the Seventh European Conference on Genetic Programming (EuroGP 2004)*, (Coimbra, Portugal, 2004), Springer: Berlin, 138-149.

70. Ohnishi, K., Adapting genotype-phenotype-mapping by using Redundant Real Representation. In *Proceedings of 2003 IEEE International Conference on Systems, Man and Cybernetics*, (Washington, DC, USA, 2003), IEEE Press, 3583-3588.

71. Panait, L., Luke, S. and Wiegand, R. Biasing Coevolutionary Search for Optimal Multiagent Behaviors. *IEEE Transactions on Evolutionary Computation*, *10* (6): 629-645, 2006.

72. Panait, L., Wiegand, R. and Luke, S., A sensitivity analysis of a cooperative coevolutionary algorithm biased for optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2004)*, (Seattle, WA, 2004), Springer: Berlin, 573-584.

73. Paredis, J., Coevolution, Memory, and Balance. In *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, (Stockholm, Sweden, 1999), Morgan Kaufman, 1212-1217.

74. Paredis, J. Coevolutionary Computation. *Artificial Life*, *2* (4): 355-375, 1995.

75. Paredis, J., Coevolving Cellular Automata: Be Aware of the Red Queen! In *Proceedings of the 7th International Conference on Genetic Algorithms (ICGA 97)*, (East Lansing, Michigan, 1997), Morgan Kaufmann, 393-400.

76. Paredis, J., The Symbiotic Evolution of Solutions and their Representations. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, (Pittsburgh, PA, 1995), Morgan Kaufmann, 359-365.

77. Paredis, J., Towards Balanced Coevolution. In *Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature*, (Paris, France, 2000), Springer, 497-506.

78. Potter, M. *The design and analysis of a computational model of cooperative coevolution*. Ph.D. Thesis, George Mason University, Fairfax, VA, 1997.

79. Potter, M. and De Jong, K. Cooperative Coevolution: An Architecture for Evolving Coadapted Subcomponents. *Evolutionary Computation*, *8* (1): 1-29, 2000.

80. Potter, M. and De Jong, K., A cooperative coevolutionary approach to function optimization. In *Proceedings of the Thrid Conference on Parallel Problem Solving from Nature*, (Jerusalem, Israel, 1994), Springer-Verlag, 249-257.

81. Potter, M., De Jong, K. and Grefenstette, J., A coevolutionary approach to learning sequential decision rules. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, (Pittsburgh, PA, 1995), Morgan Kaufmann, 366-372.

82. Sedgewick, R. *Algorithms in C++*. Addison Wesley, Boston, MA, 1992.

83. Sella, G. and Ardell, D. The Coevolution of Genes and Genetic Codes: Crick's Frozen Accident Revisted. *Journal of Molecular Evolution*, *63* (3): 297-313, 2006.

84. Shackleton, M., Shipman, R. and Ebner, M., An Investigation of Redundant Genotype-Phenotype Mappings and Their Role in Evolutionary Search. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC '00)*, (La Jolla, California, 2000), IEEE Press, 493-500.

85. Shipman, R., Genetic Redundancy: Desirable or Problematic for Evolutionary Adaptation? In *4th International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA 1999)*, (Portoroz, Slovenia, 1999), Springer-Verlag, 337-344.

86. Shipman, R. and Shackleton, M., Issues in Designing a Neutral Genotype-Phenotype Mapping (2002). In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC 2002)*, (Honolulu, HI, 2002), IEEE Press, 1360-1365.

87. Shipman, R., Shackleton, M., Ebner, M. and Watson, R., Neutral Search Spaces for Artificial Evolution: A Lesson from Life. In *Artificial Life: Proceedings of the Seventh International Conference on Artificial Life*, (Portland, OR, 2000), MIT Press, 162-169.

88. Song, D., Heywood, M. and Zincir-Heywood, A., A Linear Genetic Programming Approach to Intrusion Detection. In *Proceedings of the 2003 Genetic and Evolutionary Computation Conference (GECCO 2003)*, (Chicago, Illinois, 2003), Springer-Verlag, 2325-2336.

89. Stanley, K. and Miikkulainen, R. Evolving Neural Networks through Augmenting Topologies. *Evolutionary Computation*, *10* (2): 99-127, 2002.

90. Wagner, G. and Altenberg, L. Perspectives: Complex Adaptations and the Evolution of Evolvability. *Evolution*, *50* (3): 967-976, 1996.

91. Watson, R. and Pollack, J., Coevolutionary Dynamics in a Minimal Substrate. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, (San Francisco, California, 2001), 702-709.

92. Whigham, P. *Grammatical Bias for Evolutionary Learning (Ph.D. thesis)*. Ph.D. Thesis, University of New South Wales, Sydney, Australia, 1996.

93. Wiegand, R. and Potter, M., Robustness in Cooperative Coevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)*, (Seattle, Washington, 2006), ACM Press: New York, 369-376.

94. Wilson, G. and Heywood, M. Introducing Probabilistic Adaptive Mapping Developmental Genetic Programming with Redundant Mappings. *Genetic Programming and Evolvable Machines (Special Issue on Developmental Systems)*: (upcoming), 2007.

95. Wilson, G. and Heywood, M., Probabilistic (Genotype) Adaptive Mapping Combinations for Developmental Genetic Programming. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation (CEC 2006)*, (Vancouver, Canada, 2006), IEEE Press, 8667-8674.

96. Wilson, G. and Heywood, M., Probabilistic Adaptive Mapping Developmental Genetic Programming (PAM DGP): A New Developmental Approach. In *Proceedings of the 9th International Conference on Parallel Problem Solving from Nature (PPSN IX)*, (Reykjavik, Iceland, 2006), Springer-Verlag: Berlin, 751-760.

97. Wolberg, W. and Mangasarian, O. Multisurface method of pattern separation for medical diagnosis applied to breast cytology. *Proceedings of the National Academy of Sciences, USA, 87*: 9193-9196, 1990.

98. Wong, M. and Leung, K. Evolving recursive functions for the even-parity problem using genetic programming. In Angeline, P. and Kinnear, K., Jr. eds. *Advances in Genetic Programming II*, MIT Press, Cambridge, MA, 1996, 222-240.

99. Wong, M. and Mun, T. Evolving Recursive Programs by Using Adaptive Grammar Based Genetic Programming. *Genetic Programming and Evolvable Machines, 6*: 421-455, 2005.

100. Yu, T. *An Analysis of the Impact of Functional Programming Techniques on Genetic Programming*. Ph.D. Thesis, University College London, London, England, 1999.

101. Yu, T. Hierarchical Processing for Evolving Recursive and Modular Programs Using Higher-Order Functions and Lamda Abstraction. *Genetic Programming and Evolvable Machines, 2*: 345-380, 2001.

102. Yu, T., Polymorphism and Genetic Programming. In *Proceedings of the Fourth European Conference on Genetic Programming*, (Coimbra, Portugal, 2001), Springer-Verlag, 218-231.

103. Yu, T. and Bentley, P., Methods to Evolve Legal Phenotypes. In *Fifth International Conference on Parallel Probelm Solving from Nature*, (Amsterdam, Netherlands, 1998), Springer-Verlag, 280-291.

104. Yu, T. and Clack, C., Recursion, Lambda Abstractions and Genetic Programming. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, (Madison, Wisconsin, 1998), Morgan Kaufmann, 422-431.